

Johns Hopkins Engineering

Principles of Database Systems

Module 5 / Lecture 1
Functional Dependencies and
Normalization for Relational Databases I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Relational Database Design

- Common approaches to database design include top-down and bottom-up.
- Top-down design methodology is a more effective way than bottom-up methodology, as it captures *entity types* with their associated attributes and their *relationships* based on requirements.

Conceptual Database Design

1. Gather and analyze requirements
 2. Perform conceptual database design
 - Identify entities and key attributes such as PKs and other candidate keys
 - Identify relationship types with proper cardinalities
 - Identify and associate attributes with entity or relationship types
 - Validate conceptual model against user requirements
 - Review conceptual data model with users and other key stakeholders
- The conceptual design process may be an iterative process.

Good Relational Database Design

- A particular relation R is in “good” form to ensure the quality of intension and extension.
 - Logically grouping of attributes can form "good" relation schemas
 - Good logical design leads to good base relations
 - Good design leads good performance and scalability
- A process for reviewing and validating conceptual schema is required.

Informal Design Guidelines for Relational Schema

- Guideline 1:
 - Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.
 - Informally, each tuple in a relation should represent an entity instance or relationship instance.

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 1 Example:

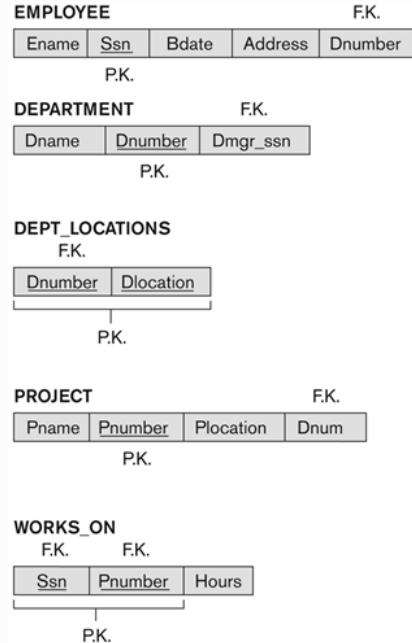


Figure 14.1 A simplified COMPANY relational database schema.

EMPLOYEE				
Ename	Ssn	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1

DEPARTMENT		
Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS	
Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON			
Ssn	Pnumber	Hours	
123456789	1	32.5	
123456789	2	7.5	
666884444	3	40.0	
453453453	1	20.0	
453453453	2	20.0	
333445555	2	10.0	
333445555	3	10.0	
333445555	10	10.0	
333445555	20	10.0	
999887777	30	30.0	
999887777	10	10.0	
987987987	10	35.0	
987987987	30	5.0	
987654321	30	20.0	
987654321	20	15.0	
888665555	20	Null	

PROJECT			
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

Figure 14.2 Sample database state for the relational schema in Figure 14.1.

Johns Hopkins Engineering

Principles of Database Systems

Module 7 / Lecture 2
Functional Dependencies and
Normalization for Relational Databases I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 2:
 - Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations.

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 2 Example:

(a)

EMP_DEPT

Ename	<u>Ssn</u>	Bdate	Address	Dnumber	Dname	Dmgr_ssn

```
graph TD; Ssn --> Bar[ ]; Bdate --> Bar[ ]; Address --> Bar[ ]; Dnumber --> Bar[ ]; Dname --> Bar[ ]; Dmgr_ssn --> Bar[ ];
```

(b)

EMP_PROJ

<u>Ssn</u>	Pnumber	Hours	Ename	Pname	Plocation
FD1					
FD2					
FD3					

```
graph TD; Ssn -- FD1 --> Pnumber[ ]; Ssn -- FD1 --> Hours[ ]; Ssn -- FD1 --> Ename[ ]; Ssn -- FD2 --> Pname[ ]; Ssn -- FD2 --> Plocation[ ]; Ename -- FD2 --> Pname[ ]; Ename -- FD2 --> Plocation[ ]; Ename -- FD3 --> Pnumber[ ]; Ename -- FD3 --> Hours[ ]; Ename -- FD3 --> Pname[ ]; Ename -- FD3 --> Plocation[ ];
```

Figure 14.3 Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 2 Example:

EMP_DEPT							Redundancy
Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn	
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555	
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555	
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321	
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321	
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555	
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555	
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321	
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555	

Figure 14.4 Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 2 Example:

For instance, a cluster is a group of tables that share the same data blocks because they share common columns as a cluster key and are often used together.

EMP_PROJ			Redundancy	Redundancy	
Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Figure 14.4 Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 2 Example:
 - Consider the relation:
EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

Update Anomaly: Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

Insert Anomaly: Cannot insert a project unless an employee is assigned to it. Inversely cannot insert an employee unless he/she is assigned to a project.

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 2 Example:
 - Consider the relation:
EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

Delete Anomaly: When a project is deleted, it will result in deleting all the employees who work on that project. Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

More storage required.

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 2 Example:
 - **Problems encountered:**
 - More data storage required
 - Insertion anomalies
 - Deletion anomalies
 - Modification anomalies
 - **Attributes of different entities (EMPLOYEES, DEPARTMENTS, PROJECTs) should *not* be mixed in the same relation**
 - **Only foreign keys should be used to refer to other entities**

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 3:
 - If possible, avoid placing attributes in a base relation when their values may frequently be null.
 - Problems encountered:
 - *May* require more data storage
 - *May* not work with aggregate operations (e.g., COUNT, SUM, AVG)
 - *May* have different interpretations

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 4:
 - Design relation schemas so that they can be JOINed with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 4 Example:

(a)

EMP_LOCS

Ename	Plocation

P.K.

Figure 14.5 Particularly poor design for the EMP_PROJ relation in Figure 14.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 14.4 onto the relations EMP_LOCS and EMP_PROJ1.

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Surgarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Surgarland
Wong, Franklin T.	Surgarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 4 Example:

EMP_PROJ1				
Ssn	Pnumber	Hours	Pname	Plocation
				P.K.

Figure 14.5 Particularly poor design for the EMP_PROJ relation in Figure 14.3(b). (a) The two relation schemas EMP_LOCS and EMP_PROJ1. (b) The result of projecting the extension of EMP_PROJ from Figure 14.4 onto the relations EMP_LOCS and EMP_PROJ1.

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

Informal Design Guidelines for Relational Schema (cont.)

■ Guideline 4 Example:

Figure 14.6 Result of applying NATURAL JOIN to the tuples in EMP_PROJ1 and EMP_LOCS of Figure 14.5 just for employee with Ssn = “123456789”. Generated spurious tuples are marked by asterisks.

EMP_LOCS * EMP_PROJ1 (Plocation)

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
*	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
*	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
*	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
*	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
*	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
*	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
*	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
*	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

*

*

*

Informal Design Guidelines for Relational Schema (cont.)

- Guideline 4:
 - Problems encountered:
 - May create spurious (false or fake) tuples
 - Bad designs for a relational database may result in erroneous results for certain JOIN operations
 - The "lossless join" property is used to guarantee meaningful results for join operations

Johns Hopkins Engineering

Principles of Database Systems

Module 7 / Lecture 3
Functional Dependencies and
Normalization for Relational Databases I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Functional Dependency

- A functional dependency (FD) describes attributes in a relation.
- Notation for Functional Dependency:
 - Let R be a relation, X and Y be arbitrary subsets of the set of attributes of R .
 - $X \rightarrow Y$ represents that X functionally determines Y , or Y is functionally dependent on X . If and only if each X value in R has precisely one Y value. In other words, if any two tuples of R have the same of X value, they also have the same value of Y .

Functional Dependency (cont.)

- For example, the value of a primary key or an alternate key in a relation can functionally determine a subset of other attributes of the relation.
 - In the EMPLOYEE relation of COMPANY ERD:
 $\{ssn\} \rightarrow \{\text{lname}, \text{fname}, \text{salary}\}$
 $\{ssn, \text{salary}\} \rightarrow \{\text{lname}, \text{fname}, \text{salary}, \text{address}\}$
 $\{\text{emp_id}\} \rightarrow \{\text{lname}, \text{fname}, \text{salary}, \text{dnumber}\}$
 - In the STUDENT relation of UNIVERSITY ERD:
 $\{\text{student_id}\} \rightarrow \{\text{lname}, \text{fname}, \text{bdate}, \text{address}\}$

Functional Dependency (cont.)

- Inference Rules for Functional Dependencies
- Six inference rules for functional dependencies
 - Let X, Y, Z and W be arbitrary subsets of the set of attributes of the given relation R, and let XY be the union of X and Y; Then:
 1. If Y is subset of X, then $X \rightarrow Y$ (Reflexive Rule)
 2. If $X \rightarrow Y$, then $XZ \rightarrow YZ$ (Augmentation Rule)
 3. If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$ (Transitive Rule)
 4. If $X \rightarrow YZ$, Then $X \rightarrow Y$ and $X \rightarrow Z$ (Decomposition Rule)
 5. If $X \rightarrow Y$, and $X \rightarrow Z$ Then $X \rightarrow YZ$ (Union Rule)
 6. If $X \rightarrow Y$, $WY \rightarrow Z$, then $XW \rightarrow Z$ (Pseudo Transitive Rule)

Functional Dependency (cont.)

- Normalization provides a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- The purpose of normalization is to review and validate relations to establish a proper set of relations that meet the data requirements.

Functional Dependency (cont.)

■ Normal Forms Based on Primary Keys

- The process of normalizing a model is one of removing all relational structures that provide multiple ways to know the same fact.
- Another way to look at normalization is as a method of controlling and eliminating redundancy in data storage.
- The other objective is to eliminate the update anomalies and lossless joins and ensure there are no spurious tuple problems.

Functional Dependency (cont.)

■ Normal Forms Based on Primary Keys

- The goal of normalization is *one fact in one place with one copy* that leads to a good database design principle.
- Data items belong together in a logical group and a group of items can be identified by its unique identifier.
- Data in group describes one, and only one, thing.

Type of Keys

- A candidate key K for an R is a subset of the set of attributes of R . It has the following properties:
 - Uniqueness: No two distinct tuples of R have the same value for K .
 - Irreducibility: No proper subset of K has the uniqueness property.
- A primary key (PK) of a relation is a unique identifier for a relation.

Type of Keys (cont.)

- A *superkey* of relation R is a set of attributes of R that includes at least one candidate key of R as a subset.

For example:

SSN is a key of EMPLOYEE relation

{SSN, LNAME, FNAME} is a superkey

{SSN, LNAME, FNAME, SALARY} is a superkey

- A candidate key is a minimal set of superkey by removing all non-key attributes.

Rolename

- A rolename is a *new name* for a foreign key. A rolename is used to indicate that the domain of the foreign key is a subset of the domain of the attribute in the parent, and performs a specific function (or role) in the entity.
- It is also used when multiple attributes occur in an entity and all are foreign keys from a parent table.
 - Example: Many STATE names may appear multiple times in an EMPLOYEE record or a STUDENT record. Rolenames should be assigned to these attributes (e.g., dob_state, current_address_state, permanent_address_state.)

Maintaining Referential Integrity

- A foreign key column value must match an existing primary key column value (or be NULL). Referential integrity constraints are specified on how referential integrity is to be maintained in a schema's DDL.
- It is necessary to specify a “Delete Constraint” to determine what should happen if a row containing a referenced primary key is deleted.

Maintaining Referential Integrity (cont.)

- It is necessary to specify an “Update Constraint” to determine what should happen if a referenced primary key is updated. This constraint only applies to the situation when the primary key is allowed to be updated.
- Options for Update Constraint and Delete Constraint are CASCADE, RESTRICTED, SET DEFAULT and NULLIFY (only if NULLs are allowed).

Johns Hopkins Engineering

Principles of Database Systems

Module 7 / Lecture 4
Functional Dependencies and
Normalization for Relational Databases I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

The Process of Normalization

- Normalization is executed as a series of steps. Each step involves a specific normal form with particular properties.
- Normalization is a process for analyzing relations based on their primary keys and functional dependencies.
- For an Unnormalized Form: A relation contains a repeating group(s) or an attribute contains a set of values.

First Normal Form (1NF)

- A relation R is in 1NF if and only if all underlying domains contain atomic values (single-valued) only, not a set of values, not repeating groups.
- Steps for converting unnormalized form to First Normal Form (1NF):
 - Remove the repeating group from the base table
 - Create a new relation with a proper new PK that includes the PK of the base relation and the repeating group

First Normal Form (1NF) (cont.)

■ Examples

(a)

DEPARTMENT			
Dname	Dnumber	Dmgr_ssn	Dlocations



(b)

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

1NF, but not a good design, why?

Another 1NF design

DEPARTMENT

Dnumber	Dname	Dmgr_ssn

DEPT_LOCATION

Dnumber	Dlocation

1NF, why this design is better?

Figure 14.9 Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

Second Normal Form (2NF)

- A relation R is in 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key (no partial dependencies).
- Every non-key attribute must be dependent upon *all parts* of the primary key of a relation.
- If each column is not dependent upon the *entire* primary key, the relation is not in 2NF.

Second Normal Form (2NF) (cont.)

- Steps for converting 2NF:
 - Determine which non-key attributes do not depend on the relation's *entire* primary key
 - Remove those attributes from the base relation
 - Create a second relation with those attributes and the attribute(s) from the PK that they are dependent upon
- Any single attribute (associated as a primary key) relation is automatically in 2NF.

Second Normal Form (2NF) (cont.)

■ Example

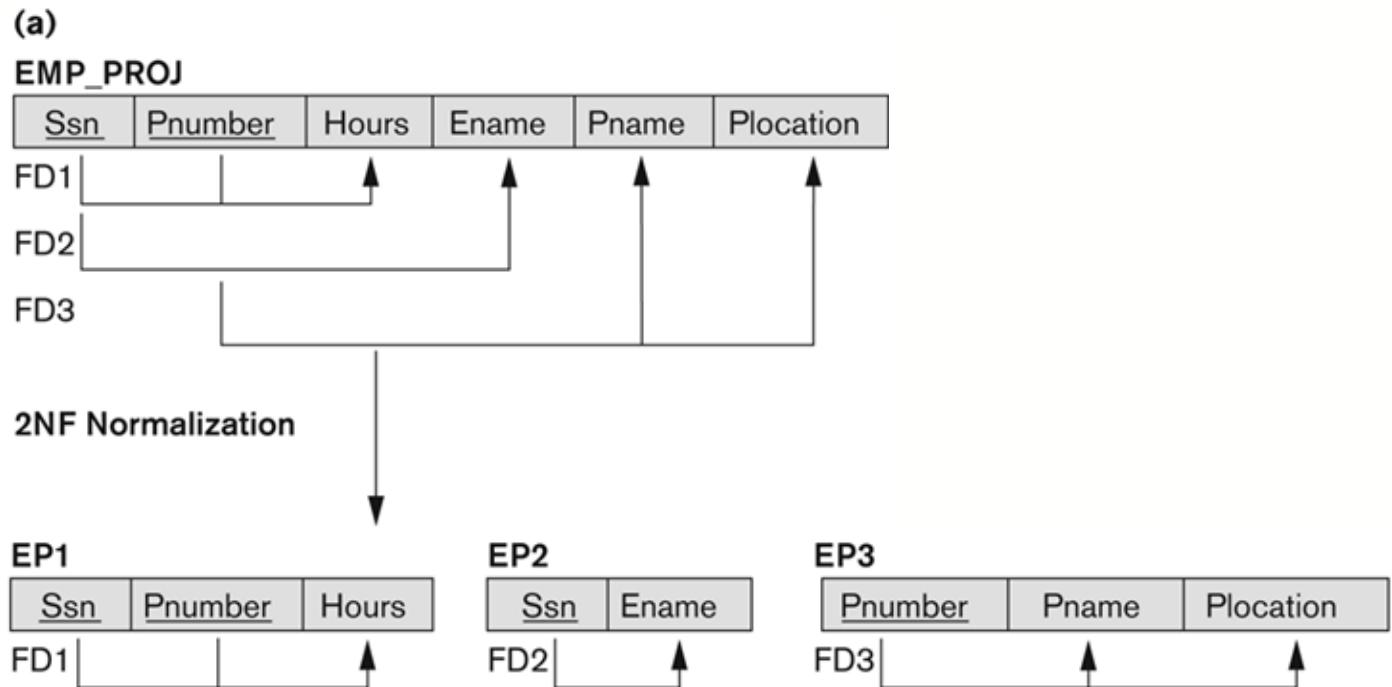


Figure 14.11 Normalizing into 2NF. (a) Normalizing EMP_PROJ into 2NF relations.

Third Normal Form (3NF)

- A relation R is in 3NF if it is in 2NF and no non-key attribute of R is functionally dependent on another non-key attribute.
- A relation R is in 3NF if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.
 - Transitive dependency

Third Normal Form (3NF) (cont.)

- Steps for converting 3NF:
 - Determine which attributes are dependent upon another non-key attribute
 - Remove those attributes from the base relation
 - Create a second relation with those attributes and the non-key attribute that they are dependent upon

Third Normal Form (3NF) (cont.)

- Example: Please describe EMP_DEPT FDs (Ssn)

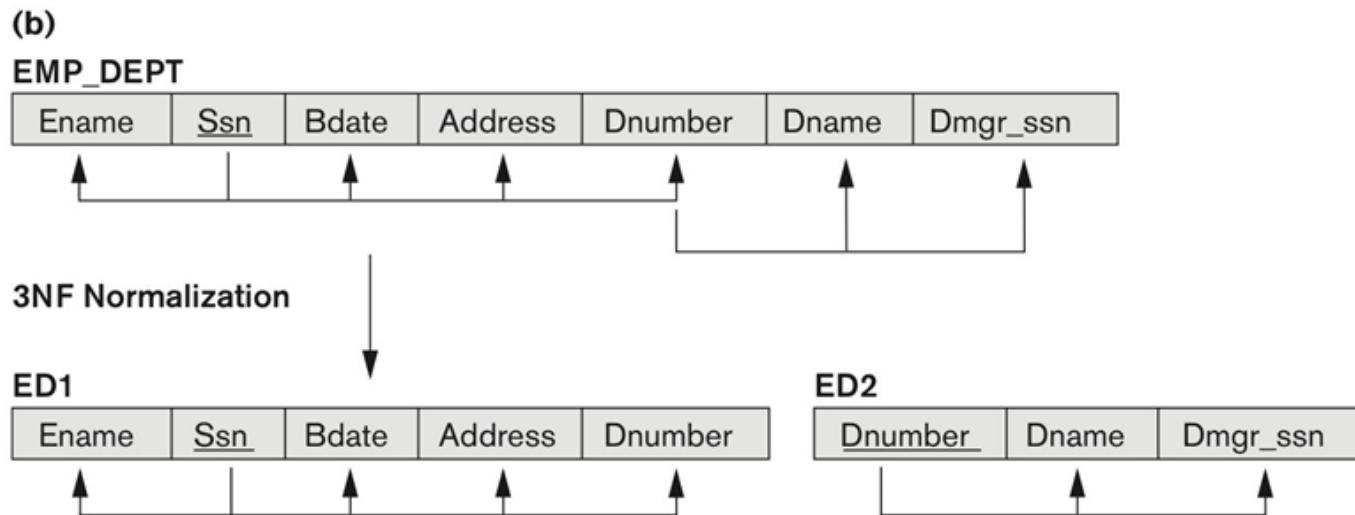


Figure 14.11 Normalizing into 3NF. (b) Normalizing EMP_DEPT into 3NF relations.

Johns Hopkins Engineering

Principles of Database Systems

Module 7 / Lecture 5
Functional Dependencies and
Normalization for Relational Databases I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Normalization Exercise

- An ORDER Relation – an unnormalized form

ORDER

order_id	order_date	customer_id	customer_name	customer_address	product_id	product_name	product_price	quantity_ordered	product_total	order_total

PK

(This order is an original entity. One order may have many products. There is a repeating group, the grey attributes, for product information.)

Normalizing into First Normal Form (1NF):

Normalizing into Second Normal Form (2NF):

Normalizing into Third Normal Form (3NF):

Normalization Exercise (cont.)

- First Normal Form (1NF)
 - Steps for converting 1NF:
 1. Remove the repeating group from the base table.
 2. Create a new table with the PK of the base table and the repeating group

ORDER

order_id	order_date	customer_id	customer_name	customer_address	order_total

PK

$\text{order_id} \rightarrow \{\text{order_date}, \text{customer_id}, \text{customer_name}, \text{customer_address}, \text{order_total}\}$

ORDER_PRODUCT

order_id	product_id	product_name	product_price	quantity_ordered	product_total

PK

$\{\text{order_id}, \text{product_id}\} \rightarrow \{\text{product_name}, \text{product_price}, \text{quantity_ordered}, \text{product_total}\}$

Normalization Exercise (cont.)

- Second Normal Form (2NF)
 - Steps for converting 2NF:
 1. Determine which non-key columns are not dependent upon the table's entire primary key.
 2. Remove those columns from the base table.
 3. Create a second table with those columns and the column(s) from the PK that they are dependent upon.

ORDER					
order_id	order_date	customer_id	customer_name	customer_address	order_total
PK					

ORDER_PRODUCT			
order_id	product_id	quantity_ordered	product_total
PK	{order_id, product_id} → {quantity_ordered, product_total}		

PRODUCT		
product_id	product_name	product_price
PK	product_id → {product_name, product_price}	

Normalization Exercise (cont.)

- Third Normal Form (3NF)
 - Steps for converting 3NF:
 1. Determine which columns are dependent upon another non-key column.
 2. Remove those columns from the base table.
 3. Create a second table with those columns and the non-key column that they are dependent upon.

ORDER			
order_id	order_date	customer_id	order_total
<u>PK</u>			

ORDER_PRODUCT			
order_id	product_id	quantity_ordered	product_total
<u>PK</u>			

CUSTOMER		
customer_id	customer_name	customer_address
<u>PK</u>		

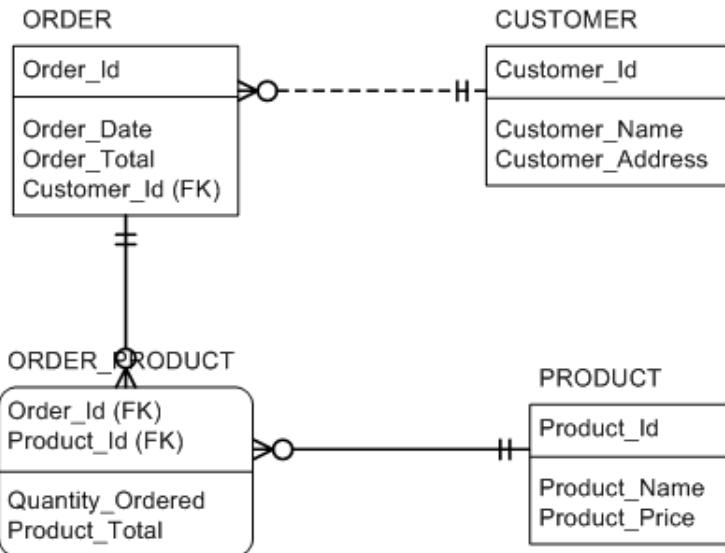
PRODUCT		
product_id	product_name	product_price
<u>PK</u>		

$\text{customer_id} \rightarrow \{\text{customer_name}, \text{customer_address}\}$

Normalization Exercise (cont.)

- Normalized ORDER in 3NF
 - Reverse the normalized relations to an ERD
 - $\text{Product_id} \rightarrow \text{Product_Price}$
 - Product_Price : a current product price associated with a product instance

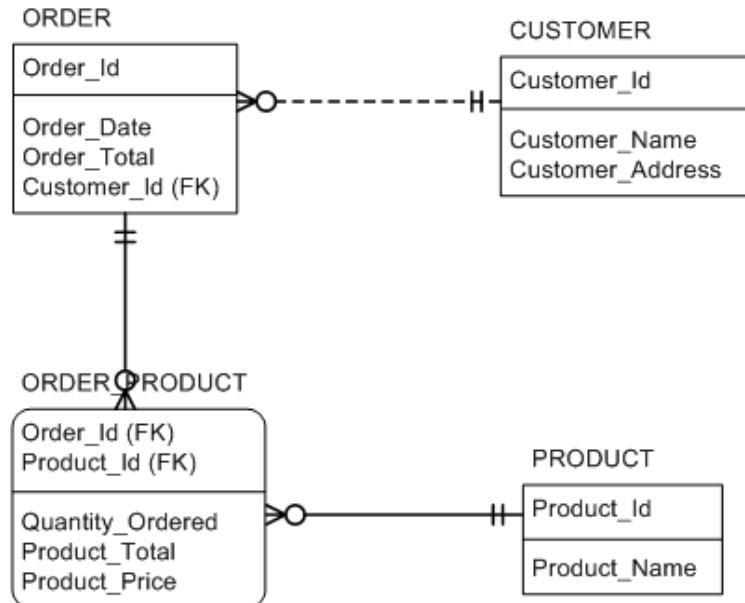
Normalized ORDER tables
Solution 1 with the assumption:
1. $\text{Product_id} \rightarrow \text{Product_Price}$



Normalization Exercise (cont.)

- Normalized ORDER in 3NF
 - Reverse the normalized relations to an ERD
 - $\{Order_id, Product_id\} \rightarrow Product_Price$
 - $Product_Price$: a price at the time of an order transaction

Normalized ORDER tables
Solution 2 with the assumption:
1. $Order_Id, Product_Id \rightarrow Product_Price$



Basic Principles When Performing Normalization

- No additional attributes will be added during the normalization process.
- It is important that you do the 1NF, 2NF, and 3NF in sequence. Otherwise, you may get the wrong answer.

Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Table 14.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Boyce/Codd Normal Form (BCNF)

- A BCNF is stricter than 3NF.
- A relation is in BCNF if and only if every determinant in the relation is a candidate key.
- When a relation contains only one candidate key, the 3NF and the BCNF are equivalent.
- If a relation has more than one candidate key, the relation may require further normalization.

Boyce/Codd Normal Form (BCNF) (cont.)

A	B	C	D
---	---	---	---

FD 1: {A, B} → {C, D}

FD 2: C → B

3NF but not in BCNF

A	C	D	C	B
---	---	---	---	---

<u>Student_id</u>	<u>Faculty_id</u>	Offering_id	Grade
-------------------	-------------------	-------------	-------

FD 1: {Student_id, Faculty_id} → {Offering_id, Grade}

FD 2: Offering_id → Faculty_id

3NF but not in BCNF (with some assumptions)

<u>Student_id</u>	<u>Offering_id</u>	Grade	<u>Offering_id</u>	<u>Faculty_id</u>
-------------------	--------------------	-------	--------------------	-------------------

Johns Hopkins Engineering

Principles of Database Systems

Module 8 / Lecture 1
Normalization for Relational Databases II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Relation Decomposition

- Universal Relation $R = \{A_1, A_2, \dots, A_n\}$
- Decomposition of R : $D = \{R_1, R_2, \dots, R_n\}$
- No attributes are lost after decomposition. In other words, each attribute in R will appear in at least one relation as *attribute preservation*.
 - $R_1 \cup R_2 \cup R_3 \dots R_n = R$

Properties of Decompositions

- **Dependence preservation property**
 - Decomposed relations preserve all original dependencies
- **Nonadditive (lossless) join property**
 - After natural joining decomposed relations, no spurious tuples are allowed.
 - Previous 2NF and 3NF examples demonstrate successive nonadditive join decomposition.

Join Loss with Null Values in Foreign Key

(a)

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnum
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	NULL

DEPARTMENT

Dname	Dnum	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

Figure 15.2a Issues with NULL-value joins. Some EMPLOYEE tuples have NULL for the join attribute Dnum.

Join Loss with Null Values in Foreign Key (cont.)

(b)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Figure 15.2b Issues with NULL-value joins. Result of applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT relations.

(c)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL	NULL	NULL
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX	NULL	NULL	NULL

Figure 15.2c Issues with NULL-value joins. Result of applying LEFT OUTER JOIN to EMPLOYEE and DEPARTMENT.

Join Loss with Null Values in Foreign Key (cont.)

- A similar problem due to the null values in the FK columns is called *dangling* records.
- The problem may occur if the design is decomposed too far.

Join Loss with Null Values in Foreign Key (cont.)

(a) EMPLOYEE_1

Ename	Ssn	Bdate	Address
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX

Figure 15.3 The dangling tuple problem. (a) The relation EMPLOYEE_1 (includes all attributes of EMPLOYEE from Figure 15.2(a) except Dnum).

Join Loss with Null Values in Foreign Key (cont.)

(b) EMPLOYEE_2

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) EMPLOYEE_3

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figure 15.3 The dangling tuple problem. (b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values). (c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).

Multivalued Dependencies (MVD)

- MVD represents a dependence between attributes (e.g., A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. The set of values for B and C are independent of each other.
- MVDs are the consequence of 1NF, which disallowed an attribute with multiple values.
- Informally, if two independent 1:M relationships are mixed in the same relation, an MVD may arise.

Fourth Normal Form (4NF)

- A relation R is in 4NF if and only if, the relation is in Boyce-Codd normal form and contains no *nontrivial multi-valued dependencies*.

Examples:

$\text{EMP}(\underline{\text{ENAME}}, \underline{\text{PNAME}}, \underline{\text{DNAME}}) \rightarrow$
 {
 $\text{EMP_PROJECT}(\underline{\text{ENAME}}, \underline{\text{PNAME}})$ and
 $\text{EMP_DEPENDENT}(\underline{\text{ENAME}}, \underline{\text{DNAME}})$
 }

$\text{EMP1}(\underline{\text{ENAME}}, \underline{\text{DEGREE}}, \underline{\text{LANGUAGE}}) \rightarrow$
 {
 $\text{EMP1_DEGREE}(\underline{\text{ENAME}}, \underline{\text{DEGREE}})$ and
 $\text{EMP1_LANGUAGE } (\underline{\text{ENAME}}, \underline{\text{LANGUAGE}})$
 }

Fourth Normal Form (4NF) (cont.)

(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

Figure 14.15 Fourth and fifth normal forms. (a) The EMP relation with two MVDs: Ename $\rightarrow\!\!>$ Pname and Ename $\rightarrow\!\!>$ Dname. (b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.

Fourth Normal Form (4NF) (cont.)

(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John
Brown	W	Jim
Brown	X	Jim
Brown	Y	Jim
Brown	Z	Jim
Brown	W	Joan
Brown	X	Joan
Brown	Y	Joan
Brown	Z	Joan
Brown	W	Bob
Brown	X	Bob
Brown	Y	Bob
Brown	Z	Bob

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y
Brown	W
Brown	X
Brown	Y
Brown	Z

EMP_DEPENDENTS

Ename	Dname
Smith	Anna
Smith	John
Brown	Jim
Brown	Joan
Brown	Bob

Figure 15.4 Decomposing a relation state of EMP that is not in 4NF. (a) EMP relation with additional tuples. (b) Two corresponding 4NF relations EMP_PROJECTS and EMP_DEPENDENTS. Decomposing a relation state of EMP that is not in 4NF.

The EMP relation is BCNF. The EMP relation with two MVDs: $Ename \rightarrow\!\!> Pname$ and $Ename \rightarrow\!\!> Dname$; and additional tuples.

Johns Hopkins Engineering

Principles of Database Systems

Module 8 / Lecture 2
Normalization for Relational Databases II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Fifth Normal Form (5NF)

- A relation R is subject to a *join dependency*
 - R can be decomposed to (R_1, R_2, \dots, R_n) and each has a subset of the attributes of R
 - R can always be recreated by joining the multiple relations (R_1, R_2, \dots, R_n)
- Relation R is in 5NF if and only if, R is in 4NF and the relation has no join dependency.

Fifth Normal Form (5NF) (cont.)

- A cyclical nature of a relation may require further normalization.

Example with an embedded three M:N relationships:

SUPPLIER_PART_PROJ(SNAME, PARTNAME, PROJNAME)

[
 SUPPLIER_PART(SNAME, PARTNAME),
 SUPPLIER_PROJ(SNAME, PROJNAME),
 PART_PROJ(PARTNAME, PROJNAME)
]

Fifth Normal Form (5NF) (cont.)

(c) SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d)

R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Figure 14.15 Fourth and fifth normal forms. (c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R_1, R_2, R_3). (d) Decomposing the relation SUPPLY into the 5NF relations R_1, R_2, R_3 .

Fifth Normal Form (5NF) (cont.)

S: Supplier, **P:** Part, **J:** Project

If $S1 \rightarrow P1$, $S1 \rightarrow J1$, $P1 \rightarrow J1$ Then
S1, P1, J1

SUPPLIER-PART-PROJECT

Supplier#	Part#	Project#
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

Legal State with
Join Dependency

SUPPLIER-PART

Supplier#	Part#
S1	P1
S1	P2
S2	P1

PART-PROJECT

Part#	Project#
P1	J2
P2	J1
P1	J1

PROJECT-SUPPLIER

Project#	Supplier#
J2	S1
J1	S1
J1	S2

Decompose

SUPPLIER-PART |X|_{Part#} PART-PROJECT

SUPPLIER-PART-PROJECT

Supplier#	Part#	Project#
S1	P1	J2
S1	P2	J1
S2	P1	J1
S2	P1	J2
S1	P1	J1

SUPPLIER-PART-PROJECT |X|_{Project#,Supplier#} PROJECT-SUPPLIER

SUPPLIER-PART-PROJECT

Supplier#	Part#	Project#
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

Spurious

Fifth Normal Form (5NF) (cont.)

Example:

- Agents represent companies. Companies represent properties. Agents sell properties.
- Mary sells RE/MAX home property and Century 21 commercial property
- Mary does not sell RE/MAX commercial property nor Century 21 homes

<u>Agent</u>	<u>Company</u>	<u>Property</u>
Mary	RE/MAX	home
Mary	Century 21	commercial property

- If an agent sells a certain property and the agent represents the company, then she sells properties for that company

<u>Agent</u>	<u>Company</u>	<u>Property</u>
Mary	RE/MAX	home
Mary	RE/MAX	commercial property
Mary	Century 21	home
Mary	Century 21	commercial property
Steve	RE/MAX	home

- Repetition of facts for Mary - Sell home twice

Fifth Normal Form (5NF) (cont.)

Example:

- No repetition of facts
- Reconstruct all true facts from 3 relations instead of the single relation.

<u>Agent</u>	<u>Company</u>
Mary	RE/MAX
Mary	Century 21
Steve	RE/MAX

<u>Company</u>	<u>Property</u>
RE/MAX	home
RE/MAX	commercial property
Century 21	home
Century 21	commercial property

<u>Agent</u>	<u>Property</u>
Mary	home
Mary	commercial property
Steve	home

Johns Hopkins Engineering

Principles of Database Systems

Module 8 / Lecture 3
Normalization for Relational Databases II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Additional Notes on Normalization

- Case tools do not understand functional dependencies (not expert systems). Therefore, they can not fully support all normal forms.
- Normalization is executed in a series of steps. As normalization proceeds, the relations become more restricted to meet the required normal forms' criteria.
- Normalization comprehensively relies on functional dependencies among key attributes and non-key attributes.

Additional Notes on Normalization (cont.)

- Each normalization process may break down a relation into more relations. How much normalization is enough?
 - 3NF is the standard. When a database is *normalized*, it generally implies that the database is in 3NF.
 - In general, a normalized design is in 3NF that may also be BCNF, 4NF, and 5NF without additional decomposition.

Additional Notes on Normalization (cont.)

- **Problems Solved by 1NF:**

- Resolve embedded multi-valued attributes and repeating groups
- Resolve embedded one-to-many relationship

- **Problems Solved by 2NF:**

- Resolve an attribute that does not depend on the FULL PK, it needs to be taken out to form a new relation
- Resolve a one-to-many identifying relationship

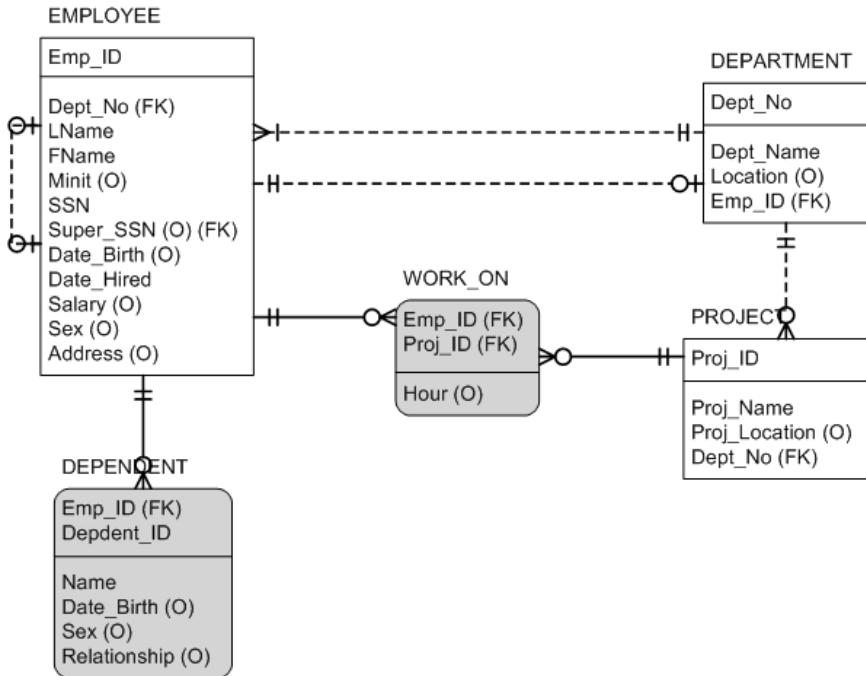
Additional Notes on Normalization (cont.)

■ Problems Solved by 3NF:

- Take the attributes that functionally depend on another non-key attribute out to form a new relation. A new parent table will be created, and the original table is a child table with a non-identifying relationship.
- Resolve a one-to-many non-identifying relationship since attributes are dependent on another non-key attribute (a PK of a parent table.)

Additional Notes on Normalization (cont.)

- Verify your ERD using normalization as a refinement process



Example: Company ERD

- Check if all underlying domains contain atomic values (single-valued) only, not a set of values, not repeating groups.
- Check for every non-key attribute is fully functionally dependent on the full primary key (no partial dependencies).
- Check for no non-key attribute of R is functionally dependent on another non-key attribute.

Additional Notes on Normalization (cont.)

- A model may be normalized, but it may still not be a correct representation of the business.
- After the normalization process, the database usually consists of more tables. There are conditions that require a database to *denormalize* in favor of performance such as quicker response time, high throughput, and high frequency for a certain set of transactions.

Additional Notes on Normalization (cont.)

- When denormalizing the database design, always start with tables in 3NF.
- A foreign key appearing twice in an entity without rolenames implies a redundant relationship structure in the model.

Performance Issues on Database Design

- It is sometimes necessary to add or change the index structure or create a cluster to improve data access time. Indexes provide quick access to rows of data and avoid full table scan. They are automatically used when referenced in the WHERE clause of a SQL statement.

Performance Issues on Database Design (cont.)

- It is good practice to build indexes for primary keys (unique indexes) and foreign keys (generally non-unique indexes).
- May be helpful to build indexes for alternate keys (unique indexes), and any non-key columns frequently used in WHERE clauses
- Consider all other options prior to denormalization, especially adding or changing the index structure.

Performance Issues on Database Design (cont.)

- Be extremely reluctant to denormalize the default design because it may cause data inconsistency problems
- Can consider denormalizing the design to reduce the number of tables and avoid the join operation to improve system performance in web applications

The Benefits of A Set of Well-designed Tables

- Reduced storage of redundant data, which eliminates the cost of updating duplicates and avoids the risk of inconsistent results based on the duplicates
- Increased ability to effectively enforce integrity constraints
- Increased ability to adapt to the growth and change of the system
- Increased productivity based on the inherent flexibility of well-designed relational systems

Role of Normalization in the Database Development Process

- A refinement process, not as an initial design process
- Intuitively group related attributes to form your entity types and relationships in ERD
- Can be done in an informal manner without the tedious process of recording functional dependencies in practice
- May identify the overlooked M-N relationships

Johns Hopkins Engineering

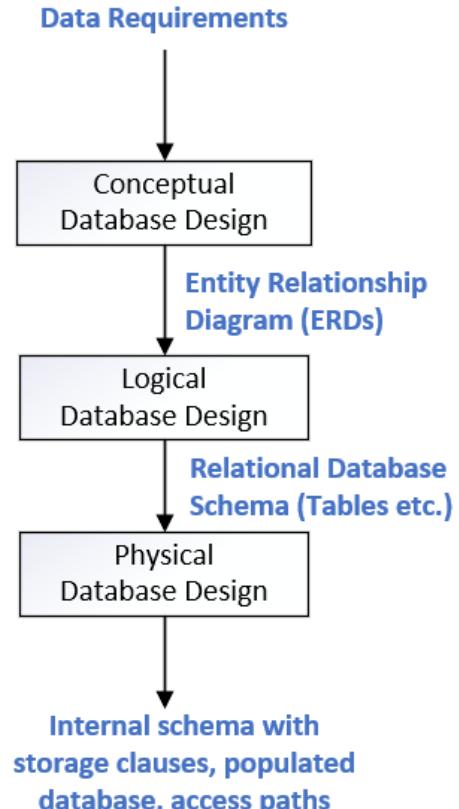
Principles of Database Systems

Module 8 / Lecture 4
ER and EER to Relational Mapping



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Database Design Process



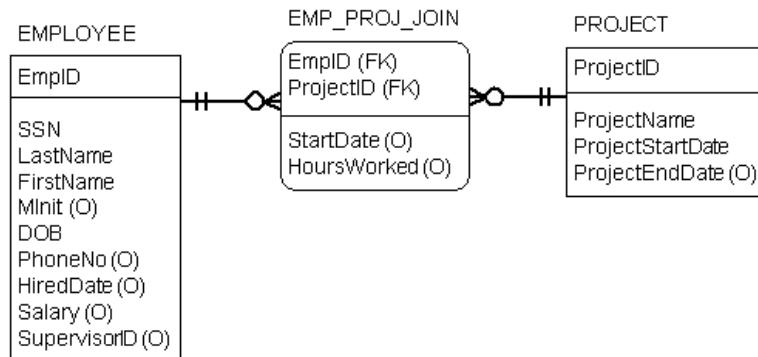
Using a conceptual schema design (Entity Relationship Diagram) to create a relational database schema

Converting ER Model to Relational Model

- Create relations for the conceptual data model to represent the entity types, relationships, and attributes that have been identified
- Implement the concepts of relational databases, primary keys, foreign keys, and data integrity

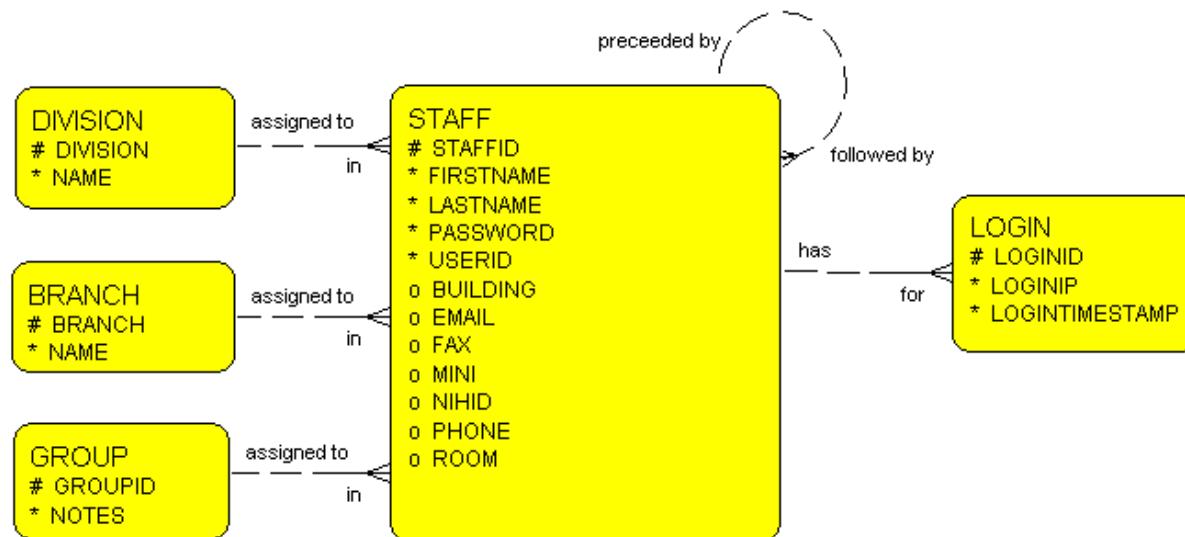
Converting ER Model to Relational Model (Cont.)

- Database design tools may have different graphical representations.
- ERwin and Visio show a foreign key attribute in a child relation:



Converting ER Model to Relational Model (Cont.)

- Oracle designer does not include a foreign key into a child relation. Why?



How to Map Entity Types and Relationships to Relations

ER Model	Mapping to Relation
Strong entity type	Create relation with all simple attributes
Weak entity type	Create relation with all simple attributes, and combine partial key of weak entity and a FK from the parent entity type as the PK
1:1 relationship type with mandatory participation on both sides	Combine entities into one relation or create two relations (see next) (e.g., EMPLOYEE vs. OFFICE or BADGE as 1-1 relationship)

How to Map Entity Types and Relationships to Relations (cont.)

ER Model	Mapping to Relation
1:1 relationship type with mandatory participation on one side	Post PK of entity on optional side to act as FK in relation representing entity on mandatory side (e.g., EMPLOYEE and CAR have 1-1 relationship)
1:M relationship type	Post PK of entity on one (parent) side to act as FK in relation representing entity on many (child) side

How to Map Entity Types and Relationships to Relations (cont.)

ER Model	Mapping to Relation
M:M relationship type	Create two 1:M relation types and follow above mapping and add additional attributes to the transitional relation (Be aware whether identifying or non-identifying relationships)
Multi-valued attribute	Create a new relation and post a copy of the PK of the parent entity into the new relation to act as a FK (e.g., DEPT_LOCATION)
N-ary relationship type	Create a new relation and post all PKs of the parent entities into the new relation to act as a PK and FKS (e.g., SUP_PRJ_PART_JOIN)

How to Map Entity Types and Relationships to Relations (cont.)

Table 9.1 Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	<i>Entity</i> relation
1:1 or 1:N relationship type	Foreign key (or <i>relationship</i> relation)
M:N relationship type	<i>Relationship</i> relation and <i>two</i> foreign keys
<i>n</i> -ary relationship type	<i>Relationship</i> relation and <i>n</i> foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Mapping COMPANY ER Schema Into A Relational Database Schema

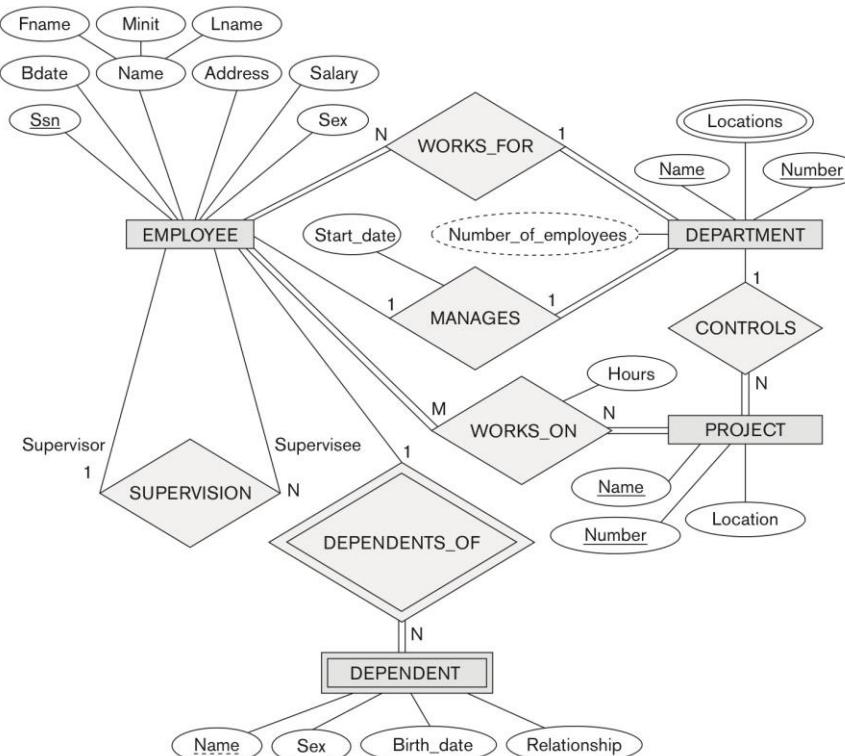


Figure 9.1 The ER conceptual schema diagram for the COMPANY database.

Mapping COMPANY ER Schema Into A Relational Database Schema (cont.)

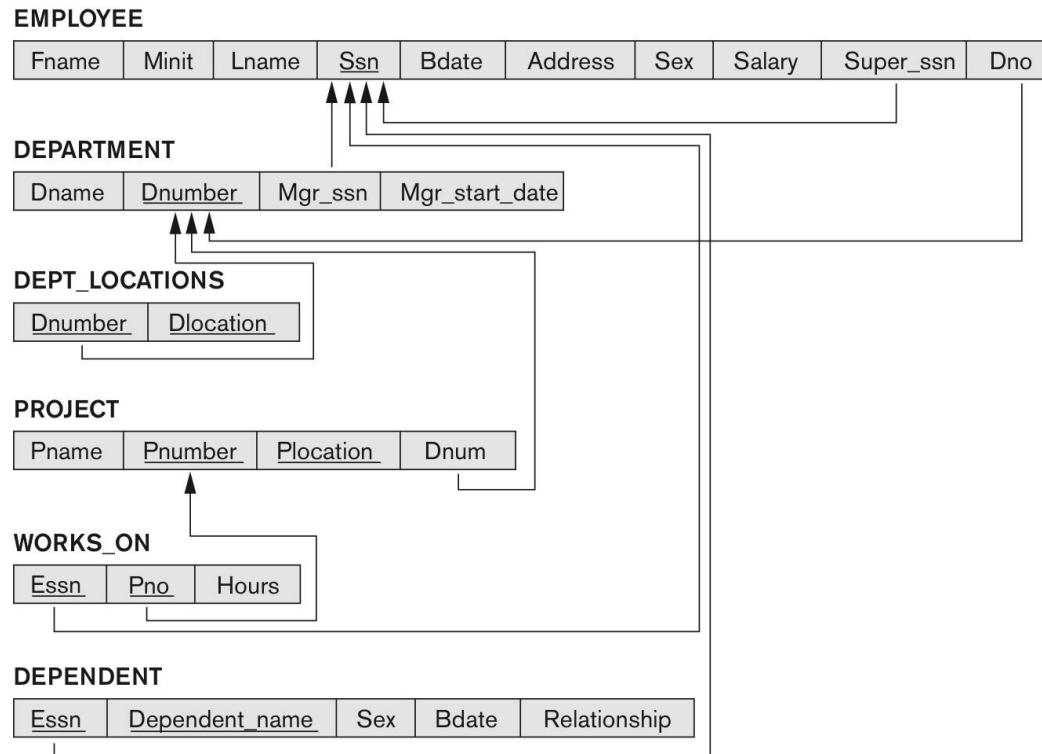
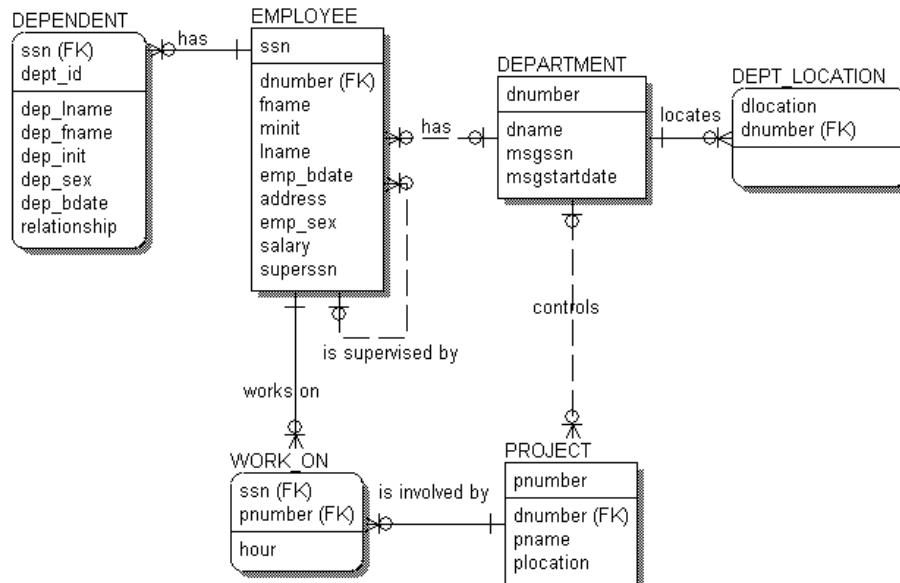


Figure 9.2 Result of mapping the COMPANY ER schema into a relational database schema.

Mapping COMPANY ER Schema Into A Relational Database Schema (cont.)

- IE notation is supported by DB design tools.

ER Diagram for the COMPANY



Mapping A Ternary Relationship Schema

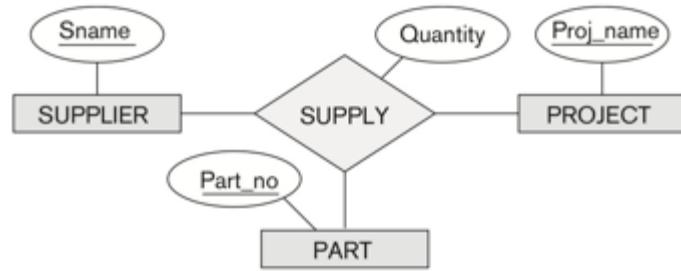


Figure 3.17 Ternary relationship types.
(a) The SUPPLY relationship.

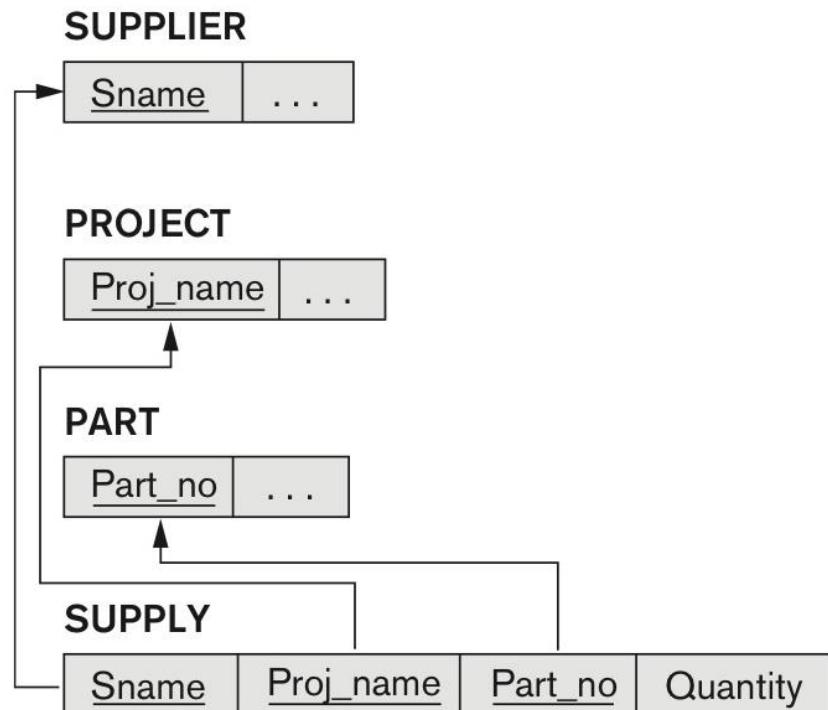


Figure 9.4 Mapping the n-ary relationship type **SUPPLY** from Figure 3.17(a).

Johns Hopkins Engineering

Principles of Database Systems

Module 8 / Lecture 5
ER and EER to Relational Mapping



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

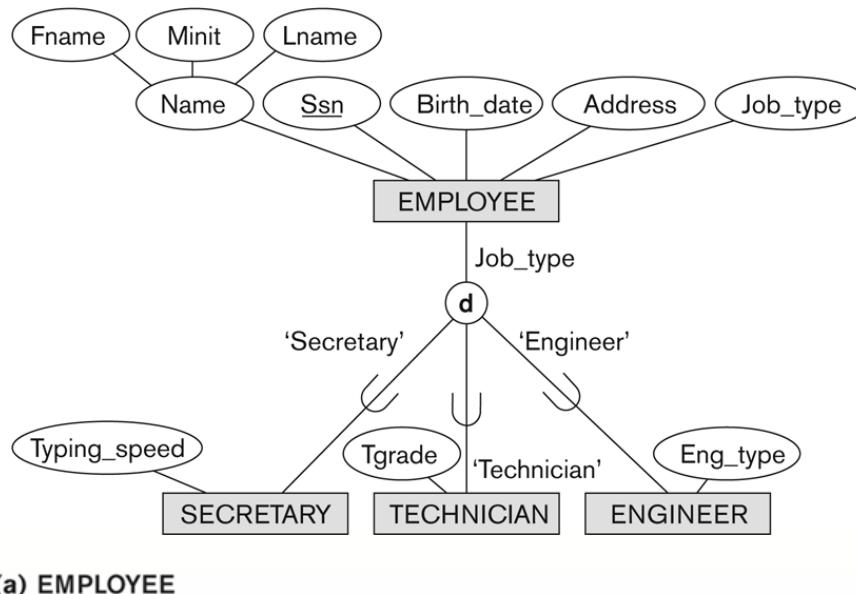
Mapping EER Model Concepts to Relations

- Convert Superclass and Subclass Relationships
 - Option 8A: Create the superclass relation and all subclass relations first, then migrate the PK from the superclass relation into each subclass relation as 1:1 relationships.
 - Multiple-relation option – superclass and subclasses Example EER schema in Figure 4.4

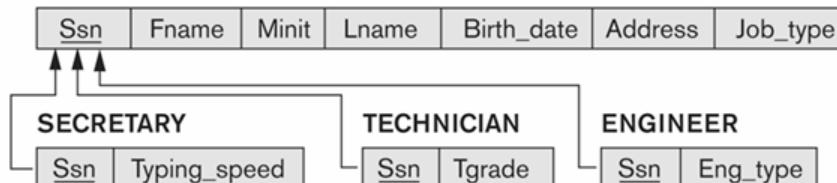
Mapping EER Model Concepts to Relations (cont.)

Figure 4.4

EER diagram notation for an attribute-defined specialization on Job_type.



(a) **EMPLOYEE**



Option 8A

Mapping EER Model Concepts to Relations (cont.)

- Convert Superclass and Subclass Relationships
 - Option 8B: Do not create a superclass relation, and create all subclass relations with all attributes from the superclass relation.
 - Multiple-relation option –subclass relations only
- Example:
- Mapping the EER schema in Figure 4.4 (b)
- EMPLOYEE → SECRETARY, TECHNICIAN, ENGINEER
- Mapping the EER schema in Figure 4.3 (b)

Mapping EER Model Concepts to Relations (cont.)

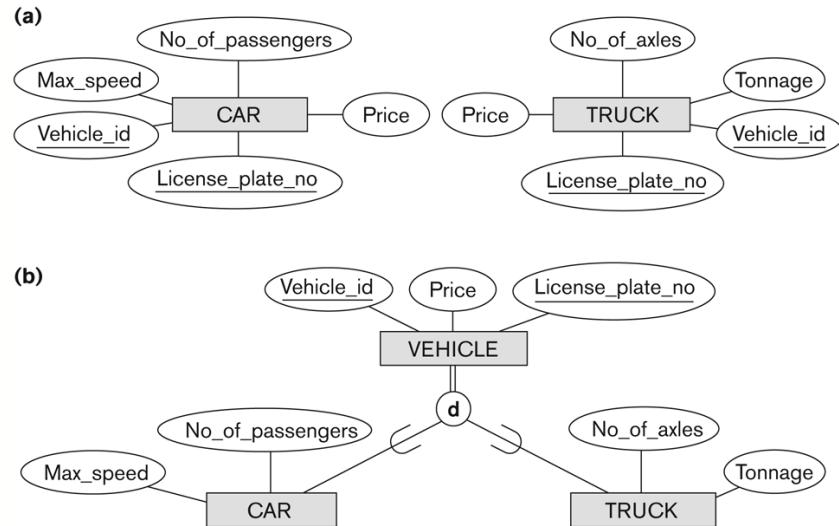


Figure 4.3

Generalization.
(a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

(b) CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers
------------	------------------	-------	-----------	------------------

TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage
------------	------------------	-------	-------------	---------

Option 8B

Mapping EER Model Concepts to Relations (cont.)

- Convert Superclass and Subclass Relationships
 - Option 8C: Create a single relation that combines (union) all attributes from all subclass relations with one type attribute (t). This approach is for a specialization whose subclasses are *disjoint*, and t is a type attribute to indicate what the tuple belongs to. Many null values will be created.
 - Single-relation option with one type attribute
- Example: Figure 4.4

(c) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type
-----	-------	-------	-------	------------	---------	----------	--------------	--------	----------

Type flag

Mapping EER Model Concepts to Relations (cont.)

- Convert Superclass and Subclass Relationships
 - Option 8D: Create a single relation that combines (union) all attributes from the superclass and all subclass relations with a set (array) of type Boolean flags to indicate whether the tuple includes/belongs to the types. This approach is for a specialization whose subclasses are *overlapping*.
 - Single-relation option with multiple type attributes Example: Mapping Figure 4.5 using option 8D with Boolean type fields Mflag and Pflag.

Mapping EER Model Concepts to Relations (cont.)

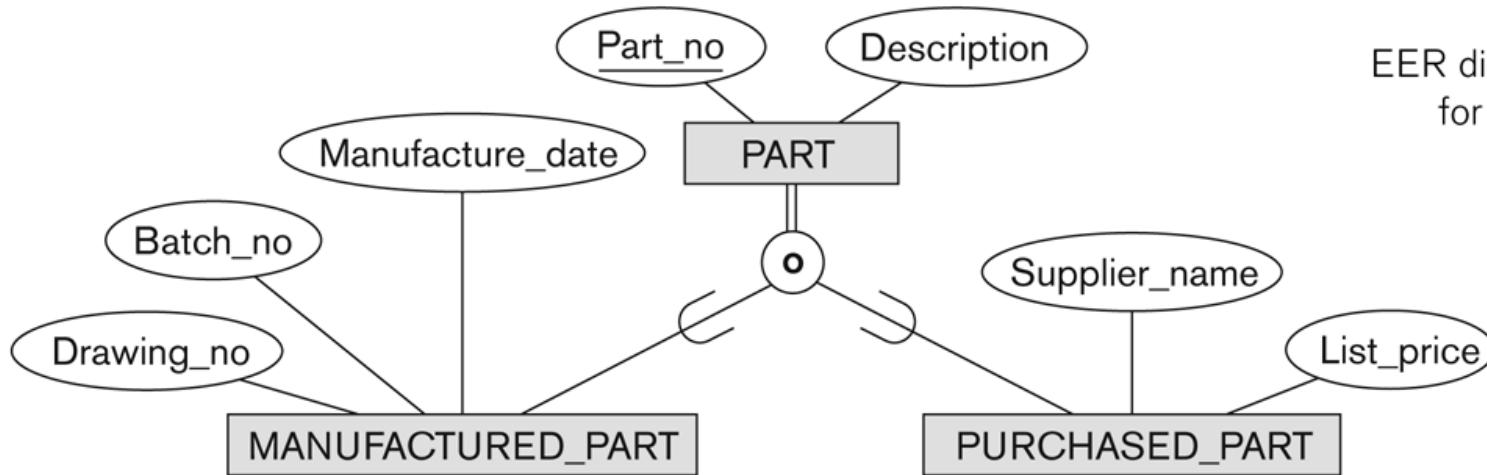


Figure 4.5
EER diagram notation
for an overlapping
(nondisjoint)
specialization.

(d) PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price
---------	-------------	-------	------------	------------------	----------	-------	---------------	------------

Option 8D

Boolean flags

Mapping Shared Subclasses

- A shared class is a subclass of several superclasses indicating multiple inheritance (**specialization lattice**)
- The classes must all have the same key attribute
- Mapping Figure 4.6

Mapping Shared Subclasses (cont.)

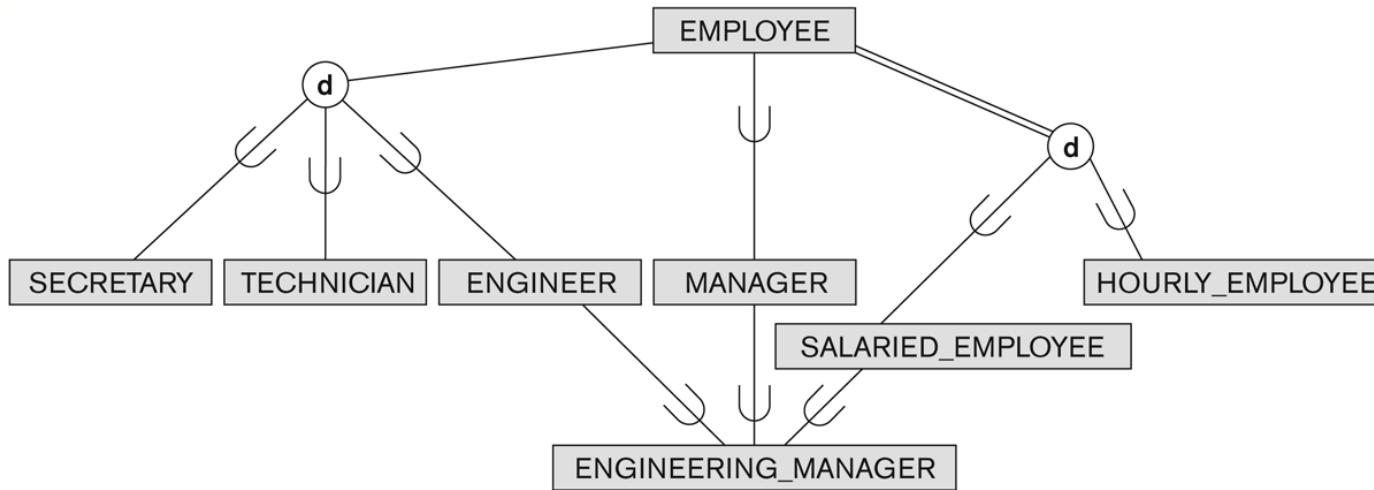


Figure 4.6

A specialization lattice with shared subclass ENGINEERING_MANAGER.

How do you map the EER specialization lattice?

Multiple-relation options (8A or 8B) in slides 15, 17

Single-relation options (8C or 8D) in slides 18, 20

Which one is a better choice and why? Open for Discussions

Mapping of Categories

- Is a subclass of the **union** of two or more superclasses that can have different entity types
- Can use a surrogate key

Example: Mapping the EER categories (union types) in Figure 4.8 to relations.

Mapping of Categories (cont.)

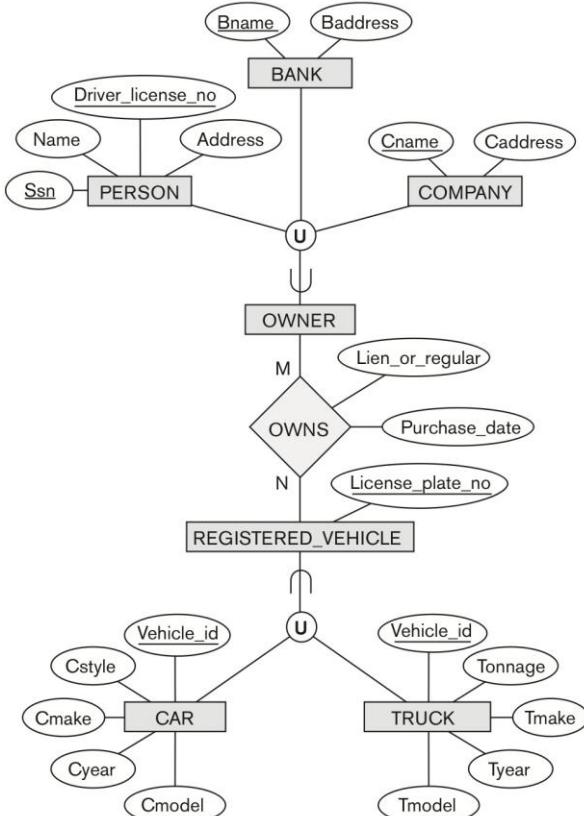


Figure 4.8 Two categories (union types): OWNER and REGISTERED_VEHICLE.

Mapping of Categories (cont.)

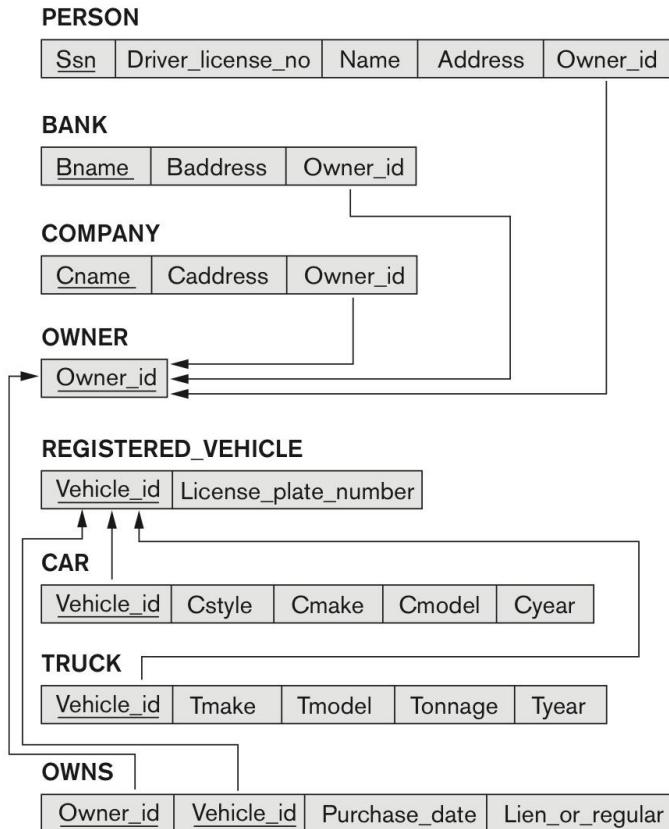


Figure 9.7 Mapping the EER categories (union types) in Figure 4.8 to relations.

Challenges on EER-to-Relation Mapping

- Can have multiple options available for specialization and generalization
 - Create more relations (tables) with multiple-relation options
 - Create fewer relations (tables) with single-relation options
- Consider implementation complications and performance when considering EER-to-Relation mapping

EER-to-Relation Mapping using ERwin

- Use supertype and subtype instead of superclass and subclass
- Create an identifying relationship between a supertype entity and its subtype entities
- Apply a transform to create an identifying relationship between a supertype entity and its subtype entities
 - Create a simple model
 - Improve query performance
 - Simplify application development and maintenance

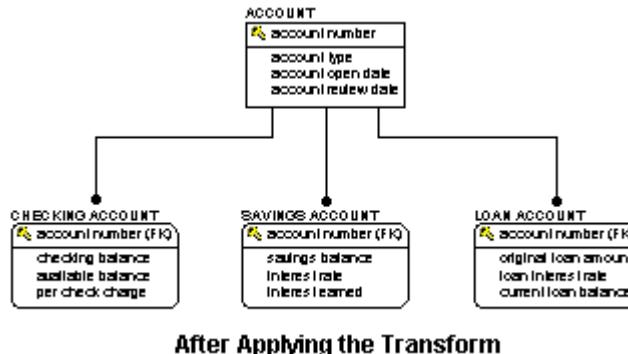
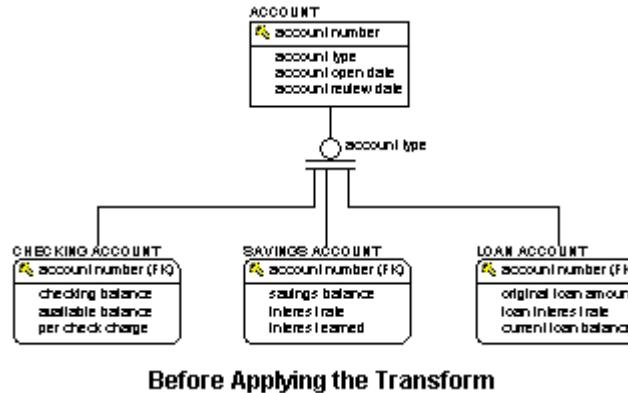
EER-to-Relation Mapping using ERwin (cont.)

Example:

Bank ERD with various account types:
Checking, Saving and Loan

Mapping multiple relations with supertype and subtypes

Comments on this design



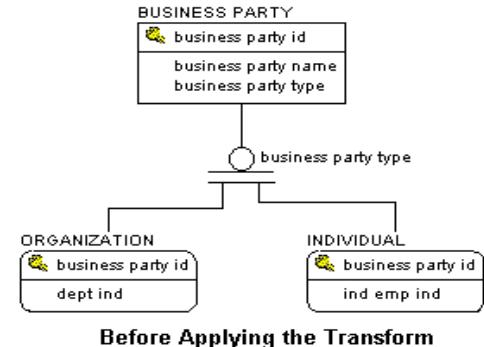
EER-to-Relation Mapping using ERwin (cont.)

Example:

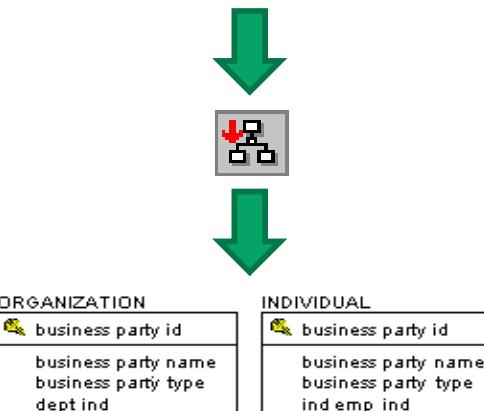
Business Party ERD with Organization and individual types

Mapping to multiple relations with a result of “Rolling Down” two subtype relations

Comments on this design



Before Applying the Transform



After Applying the Transform

EER-to-Relation Mapping using ERwin (cont.)

Example:

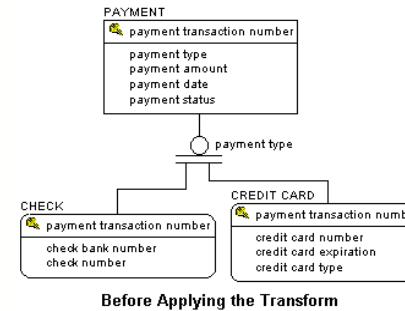
Payment ERD with Check and Credit Card types

Mapping to multiple relations with a result of “Rolling up” a supertype relation

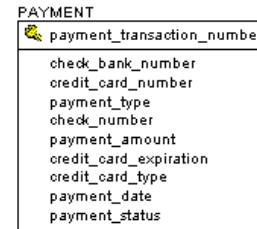
Comments on this design

Relational Mapping Considerations

- Normalized and Compact Design



Before Applying the Transform



After Applying the Transform

Johns Hopkins Engineering

Principles of Database Systems

Module 9 / Lecture 1
Structured Query Language (SQL)
The Relational DB Language I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Structured Query Language History

- The relational database model was originally developed by Dr. Codd. The idea was published in 1970 to become one of the greatest research papers in computer history.
- Dr. E(dger). F. "Ted" Codd is one of the great names in computing. He is often called the Father of Relational Databases.

Structured Query Language History (cont.)

- Colleagues at IBM and Berkeley experimented with his ideas and found that they simplified queries that were complex or too tied to the internal structures in other database paradigms.
- The Structured Query Language (SQL) language was originally developed by IBM in a prototype RDBMS in the mid-1970. SQL has also been implemented in IBM's DB2.

Structured Query Language History (cont.)

- The original commercial SQL language was introduced by Oracle Cooperation in 1979.
- The first standard SQL1 (or SQL 86) was introduced by ANSI in 1986. The second standard SQL2 (or SQL 92) was introduced by ISO in 1992. The third standard SQL3 with object-oriented and other new features was introduced by ISO in 1999.
- SQL 2003 includes XML support, sequences, columns with auto-generated values, and MERGE (upsert). SQL 2006 includes more XML features and XQuery.
- SQL 2008 includes user-defined types and routines, reference type, collection types, triggers, BLOBs, and CLOBs, TRUNCATE, INSTEAD OF triggers and others.
- SQL 2011 adds temporal data and FETCH clause.
- SQL 2016 adds JSON, polymorphic table functions, and row pattern matching.

Benefits for Using SQL

- SQL processes sets of records rather than just one at a time.
- SQL does not require you to specify the access method to the data. This feature makes it easier for you to concentrate on obtaining the desired data.
- It acquires the desired results without specifying an access method to the data.
- RDBMS uses an optimizer for determining the fastest means of accessing the specified data.

Benefits for Using SQL (cont.)

- Can be used by all users. SQL provides easy-to-learn commands that are both consistent and applicable for all users.
- Performs commands for a variety of tasks:
 - Data definition
 - Data manipulation
 - Data retrieval
- Is supported by all major relational databases. All programs written in standard SQL are portable.

Programming Paradigms

- Structured programming
- Procedural programming
- Modular programming
- Data abstraction
- Object-oriented programming

Programming Paradigms (cont.)

- SQL is designed for a specific, limited purpose – DDL, DML, and DQL (querying data) in a relational database.
- SQL is a case insensitive language. A string constant (e.g. 'Smith') is case sensitive.
- Different RDBMS's may use different syntax and support various subsets of SQL standards.
- Standard syntax ensures code portability. Vendor's specific features/syntax are not portable.

Johns Hopkins Engineering

Principles of Database Systems

Module 9 / Lecture 2
Structured Query Language (SQL)
The Relational DB Language I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

SQL Data Definition Language (DDL)

- DDL commands (e.g., CREATE, ALTER, DROP) are automatically committed while DML commands may need an explicit commit statement.
- The order for creating tables may be important because the DBMS verifies the referential integrity constraints.

SQL Data Definition Language (cont.)

- Oracle is used for a SQL demo. A sample database with two tables “emp” and “dept”. The database is from an Oracle default instance ORCL.

EMP								
empno	ename	title	mgr	hire_date	salary	comm	deptno	
7369	SMITH	CLERK	7902	12/17/1980	800		20	
7499	ALLEN	SALESMAN	7698	2/20/1981	1600	300	30	
7521	WARD	SALESMAN	7698	2/22/1981	1250	500	30	
7566	JONES	MANAGER	7839	4/2/1981	2975		20	
7654	MARTIN	SALESMAN	7698	9/28/1981	1250	1400	30	
7698	BLAKE	MANAGER	7839	5/1/1981	2850		30	
7782	CLARK	MANAGER	7839	6/9/1981	2450		10	
7788	SCOTT	ANALYST	7566	4/19/1987	3000		20	
7839	KING	PRESIDENT		11/17/1981	5000		10	
7844	TURNER	SALESMAN	7698	9/8/1981	1500	0	30	
7876	ADAMS	CLERK	7788	5/23/1987	1100		20	
7900	JAMES	CLERK	7698	12/3/1981	950		30	
7902	FORD	ANALYST	7566	12/3/1981	3000		20	
7934	MILLER	CLERK	7782	1/23/1982	1300		10	

DEPT		
deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SQL Data Definition Language (cont.)

- To create multiple tables and views and other database objects (not an instance of an object, or a row of a table) in a *single transaction* use CREATE SCHEMA.

Example:

```
CREATE SCHEMA db_proj AUTHORIZATION jdoe
```

- Database objects such as tables, views, schemas are associated with a database USER. Therefore, a user (e.g., jdoe) has to be created before creating the schema.

SQL Data Definition Language (cont.)

- RDBMS uses the catalog in which descriptions of data items are stored and the catalog is accessible to users.

```
SQL> SELECT * FROM CAT;  
TABLE_NAME          TABLE_TYPE  
-----  
DEPT                TABLE  
EMP                 TABLE
```

- The “DESCRIBE” command returns the definitions of tables and views.

```
SQL> DESCRIBE dept;  
NAME      Null?    TYPE  
-----  
DEPTNO   NOT NULL NUMBER(2)  
DNAME        VARCHAR2(20)  
LOC         VARCHAR2(30)
```

SQL – CREATE TABLE

- Purpose: To create a *table*, the basic structure to hold user data, specifying the following information:
 - Column definitions
 - Integrity constraints (different approaches)
 - Others – storage characteristics, tablespace, cluster, degree of parallelism used to create the table and the default degree of parallelism for queries on the table (Oracle features)
- SQL 92 Summary document for reference

SQL – CREATE TABLE (cont.)

■ CREATE TABLE Syntax

CREATE TABLE TableName
(<Column-Definition>* [, <Table-Constraint>*])

<Column-Definition>: ColumnName DataType
[DEFAULT { DefaultValue | USER | NULL }]
[[CONSTRAINT ConstraintName] NOT NULL]
[[CONSTRAINT ConstraintName] UNIQUE]
[[CONSTRAINT ConstraintName] PRIMARY KEY]
[[CONSTRAINT ConstraintName] FOREIGN KEY REFERENCES TableName
[(ColumnName)] [ON DELETE <Action-Specification>]
[ON UPDATE <Action-Specification>]]

The asterisk * after a syntax element indicates that a comma-separated list can be used.
Names enclosed in angle brackets <> denote definitions defined later in the syntax.
Square brackets [] enclose optional elements.
Curly brackets {} enclose choice elements.
The parentheses () denote themselves.
Double hyphens -- denote comments that are not part of the syntax.

SQL – CREATE TABLE (cont.)

- CREATE TABLE Syntax

CREATE TABLE TableName
(<Column-Definition>* [, <Table-Constraint>*])

<Table-Constraint>: [CONSTRAINT ConstraintName]

{ <Primary-Key-Constraint> |

<Foreign-Key-Constraint> |

<Uniqueness-Constraint> |

<Primary-Key-Constraint>: PRIMARY KEY (ColumnName*)

<Foreign-Key-Constraint> FOREIGN KEY (ColumnName*)

 REFERENCES TableName (ColumnName*)

 [ON DELETE <Action-Specification>]

 [ON UPDATE <Action-Specification>]

<Uniqueness-Constraint>: UNIQUE (ColumnName*)

<Action-Specification>: { CASCADE | SET NULL | SET DEFAULT | NO ACTION }

SQL – CREATE TABLE (cont.)

A sample database with two tables “**emp**” and “**dept**” in an Oracle default instance ORCL. Table with *column constraints* to define the “**emp**” table owned by “SCOTT”:

```
CREATE TABLE scott.emp
(empno NUMBER          CONSTRAINT pk_emp PRIMARY KEY,
ename VARCHAR2(10)    CONSTRAINT nn_ename NOT NULL
                      CONSTRAINT upper_ename
                      CHECK (ename = UPPER(ename)),
job VARCHAR2(9),
mgr NUMBER            CONSTRAINT fk_mgr REFERENCES scott.emp(empno),
hiredate DATE         DEFAULT SYSDATE,
sal NUMBER(10,2)       CONSTRAINT ck_sal CHECK (sal > 500),
comm NUMBER(9,0)        DEFAULT NULL,
deptno NUMBER(2)       CONSTRAINT nn_deptno NOT NULL
                      CONSTRAINT fk_deptno REFERENCES scott.dept(deptno))
```

SQL – CREATE TABLE (cont.)

Table with table constraints to define the “emp_1” table owned by SCOTT:

```
CREATE TABLE scott.emp_1
(empno NUMBER           NOT NULL,
ename VARCHAR2(10)      NOT NULL,
job  VARCHAR2(9),
mgr   NUMBER,
hiredate DATE            DEFAULT SYSDATE,
sal   NUMBER(10,2)        CONSTRAINT ck_sal_1 CHECK (sal > 500),
comm  NUMBER(9,0)         DEFAULT NULL,
deptno NUMBER(2)         NOT NULL,
CONSTRAINT pk_emp PRIMARY KEY(empno),
CONSTRAINT fk_deptno_1 FOREIGN KEY(deptno) REFERENCES scott.dept(deptno),
CONSTRAINT fk_mgr FOREIGN KEY(mgr) REFERENCES scott.emp(empno) )
```

SQL – ALTER TABLE

- ALTER TABLE Syntax

```
ALTER TABLE TableName
```

```
{ ADD { <Column-Definition> | , <Table-Constraint> } |  
    ALTER ColumnName { SET DEFAULT DefaultValue | DROP DEFAULT } |  
    DROP ColumnName { CASCADE | RESTRICT } |  
    DROP CONSTRAINT ConstraintName { CASCADE | RESTRICT } }
```

SQL – ALTER TABLE (cont.)

- Purpose: To alter the definition of a table in one of the following ways:
 - Add or delete a column
 - Add an integrity constraint
 - Redefine a column (datatype, size, default value)
 - Enable, disable, or drop an integrity constraint or trigger
 - Modify storage characteristics or other parameters, explicitly allocate an extent, explicitly de-allocate the unused space of a table, allow or disallow writing to a table, modify the degree of parallelism for a table (Oracle features)

SQL – ALTER TABLE (cont.)

Using CREATE TABLE to create “emp” table without specifying PK and FKS, and using ALTER TABLE to add PK and FKS:

```
CREATE TABLE emp
(empno NUMBER NOT NULL,
ename VARCHAR2(10) NOT NULL,
job VARCHAR2(9),
mgr NUMBER,
hiredate DATE      DEFAULT SYSDATE,
sal NUMBER(10,2)   CONSTRAINT ck_sal CHECK (sal > 500),
comm NUMBER(9,0)   DEFAULT NULL,
deptno NUMBER(2) NOT NULL);
```

```
ALTER TABLE scott.emp
ADD (CONSTRAINT pk_emp PRIMARY KEY(empno),
CONSTRAINT fk_deptno FOREIGN KEY(deptno) REFERENCES scott.dept(deptno),
CONSTRAINT fk_mgr FOREIGN KEY(mgr) REFERENCES scott.emp(empno) )
```

Johns Hopkins Engineering

Principles of Database Systems

Module 9 / Lecture 3
Structured Query Language (SQL)
The Relational DB Language I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Types of Constraints in SQL

- Types of constraints are **C** (CHECK constraint for a domain and NOT NULL constraint for required data), **P** (PRIMARY KEY), **R** (FOREIGN KEY), and **U** (UNIQUE KEY).
 - Example: Use Oracle to confirm constraints on a table with the `USER_CONSTRAINTS` data dictionary table:

SQL> DESCRIBE user_constraints		
Name	Null?	Type
OWNER	NOT NULL	VARCHAR2 (30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2 (30)
CONSTRAINT_TYPE		VARCHAR2 (1)
TABLE_NAME	NOT NULL	VARCHAR2 (30)

Types of Constraints in SQL (cont.)

- Example: Using SQL to view table's constraints associated with EMP3

```
SQL> SELECT owner, constraint_name, constraint_type, table_name
  2  FROM user_constraints
  3  WHERE table_name = 'EMP3';
```

OWNER	CONSTRAINT_NAME	C TABLE_NAME
SCOTT	SYS_C00997	C EMP3
SCOTT	SYS_C00998	C EMP3
SCOTT	SYS_C00999	C EMP3
SCOTT	CK_SAL_3	C EMP3
SCOTT	PK_EMP3	P EMP3
SCOTT	FK_DEPTNO3	R EMP3
SCOTT	FK_MGR3	R EMP3

```
7 rows selected.
```

Domain in SQL

- In a column constraint, the CHECK clause can reference a domain constraint or can be defined explicitly.

```
CREATE DOMAIN DomainName AS DataType  
[ DEFAULT defaultValue ]  
[ CHECK (SearchCondition) ] ;
```

Example:

```
sex CHAR CHECK (sex in ('M', 'F')) ,
```

```
CREATE DOMAIN SexType AS CHAR  
DEFAULT 'M'  
CHECK ( VALUE IN ('M', 'F') ) ;
```

DROP and TRUNCATE in SQL

- Removing a table, schema, and other database objects:

DROP TABLE TableName { CASCADE | RESTRICT }

DROP SCHEMA SchemaName { CASCADE | RESTRICT }

- Common database objects – SCHEMA, TABLE, INDEX, VIEW, DOMAIN. *Use **DROP** carefully!*
- Quickly removing all records in a table, and the data may not be able to recover. RDBMS implements differently.

TRUNCATE TABLE TableName

Data Type in SQL

■ ISO Data Type

ISO Data Type	Declarations
Boolean	BOOLEAN
Character	CHAR
Bit	BIT
Exact Numeric	NUMERIC, DECIMAL, INTEGER
Approximate Numeric	FLOAT, REAL, DOUBLE PRECISION
DateTime	DATE, TIME, TIMESTAMP
Large objects	BLOB (Binary Large Object)
Interval	INTERVAL

Check your RDBMS for the supported data types

Data Type in SQL (cont.)

- Data type and length are the most fundamental integrity constraints applied to data in a database.
- Common practice for data types:
 - Boolean – True or False
 - Exact Numeric – SMALLINT, INTEGER, or DECIMAL
 - Approximate Numeric – FLOAT or REAL
 - Character – CHAR or VARCHAR
 - Present different characteristics
 - Date and time – DATE, TIME, or TIMESTAMP
 - Allow arithmetic calculation and build-in functions
 - Is “10302007” good or bad?
 - Large Objects – BLOB, CLOB, DBCLOB, GRAPHIC, VARGRAPHIC (DBMS dependent)

Query in SQL

■ SELECT Syntax:

```
SELECT [ DISTINCT ] <Column-Specification>*
FROM <Table-Specification>
[ WHERE <Row-Condition> ]
[ GROUP BY ColumnName* ]
[ HAVING <Group-Condition> ]
```

- The **SELECT**-clause lists the attributes or functions to be retrieved
- The **FROM**-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries

Query in SQL (cont.)

- **SELECT Syntax:**
 - The **WHERE** clause specifies the conditions for selection and join of records from the tables specified in the **FROM** clause
 - **GROUP BY** specifies grouping attributes
 - **HAVING** specifies a condition for selection of groups
 - **ORDER BY** specifies an order for displaying the result of a query

Query in SQL (cont.)

■ Comparison Operators

- $=$ Equal to
- $\textcolor{red}{<>}$ Not equal to (ISO standard; $\textcolor{black}{\neq}$ may work for some RDBMS)
- $>$ Greater than
- $\textcolor{red}{\geq}$ Greater than or equal to
- $<$ Less than
- $\textcolor{red}{\leq}$ Less than or equal to

Query in SQL (cont.)

- Logical Operator Precedence
 - The logical operators and arithmetic operators in SQL are handled according to precedence rules.
 - These operators can affect the SQL evaluation of an expression in subtle and unexpected ways in your results.

Query in SQL (cont.)

- Logical Operator Precedence (cont.)
 - The precedence hierarchy is:
Parentheses (highest)
Multiplication, Division
Subtraction/Addition
NOT
AND
OR (lowest)
 - The parentheses operators can be used to ensure the correct evaluation sequence for the SQL statements.

Johns Hopkins Engineering

Principles of Database Systems

Module 9 / Lecture 4

Structured Query Language (SQL)

The Relational DB Language I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

SQL Query Examples

- List all employees and departments.

```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
14 rows selected.
```

```
SQL> SELECT * FROM dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
4 rows selected.
```

SQL Query Examples (cont.)

- List all jobs for all the employees
- List unique jobs for all the employees

```
SQL> SELECT job FROM emp;
```

```
JOB  
-----  
CLERK  
SALESMAN  
SALESMAN  
MANAGER  
SALESMAN  
MANAGER  
MANAGER  
ANALYST  
PRESIDENT  
SALESMAN  
CLERK  
CLERK  
ANALYST  
CLERK  
14 rows selected.
```

```
SQL> SELECT DISTINCT job FROM emp;
```

```
JOB  
-----  
ANALYST  
CLERK  
MANAGER  
PRESIDENT  
SALESMAN  
5 rows selected.
```

SQL Query Examples (cont.)

- List all employees from department 10

```
SQL> SELECT *
  2  FROM emp
  3  WHERE deptno = 10;
EMPNO ENAME      JOB          MGR HIREDATE        SAL       COMM  DEPTNO
----- -----
 7782 CLARK      MANAGER     7839 09-JUN-81    2450
 7839 KING       PRESIDENT   17-NOV-81    5000
 7934 MILLER     CLERK      7782 23-JAN-82    1300
3 rows selected.
```

- List the name, salary and job of all employees in department 20 whose salary is more than \$2,000

```
SQL> SELECT ename, sal, job
  2  FROM emp
  3  WHERE deptno = 20
  4  AND sal > 2000;
ENAME          SAL JOB
----- -----
JONES          2975 MANAGER
SCOTT          3000 ANALYST
FORD           3000 ANALYST
3 rows selected.
```

SQL Query Examples (cont.)

- List the name, job and salary of all employees whose job is manager or president

```
SQL> SELECT ename, job, sal
  2  FROM emp
  3 WHERE job = 'MANAGER' OR job = 'PRESIDENT';
    ENAME      JOB          SAL
  -----
  JONES      MANAGER      2975
  BLAKE      MANAGER      2850
  CLARK      MANAGER      2450
  KING       PRESIDENT    5000
  4 rows selected.
```

- List all employees whose job is manager or is a clerk in department 10

```
SQL> SELECT empno, ename, job, deptno
  2  FROM emp
  3 WHERE job = 'MANAGER' OR job = 'CLERK' AND deptno = 10;
    EMPNO ENAME      JOB          DEPTNO
  -----
  7566  JONES      MANAGER      20
  7698  BLAKE      MANAGER      30
  7782  CLARK      MANAGER      10
  7934  MILLER     CLERK       10
  4 rows selected.
```

SQL Query Examples (cont.)

- List the employees who work in department 10 and whose job is either manager or clerk

```
SQL> SELECT empno, ename, job, deptno
  2  FROM emp
  3 WHERE (job = 'MANAGER' OR job = 'CLERK') AND deptno = 10;
    EMPNO ENAME      JOB          DEPTNO
----- 
  7782 CLARK       MANAGER      10
  7934 MILLER     CLERK       10
2 rows selected.
```

- List all employees whose salary is between \$1,200 and \$1,400 inclusive

```
SQL> SELECT ename, job, sal
  2  FROM emp
  3 WHERE sal BETWEEN 1200 AND 1400;
    ENAME      JOB          SAL
----- 
  WARD       SALESMAN    1250
  MARTIN    SALESMAN    1250
  MILLER    CLERK       1300
3 rows selected.
```

Pattern Matching in SQL

- “LIKE” for pattern matching to specify a search condition in WHERE clause
 - **%** for any sequence of characters as a wild card character
 - **_** for any single character
 - Not good for full-text search in long text attributes or documents

Pattern Matching in SQL (cont.)

- List employees whose name begins with an 'M'

```
SQL> SELECT ename, job, deptno
  2  FROM emp
  3  WHERE ename LIKE 'M%';
ENAME      JOB          DEPTNO
-----  -----
MARTIN    SALESMAN        30
MILLER    CLERK           10
2 rows selected.
```

- List employees whose name has an 'R' as the third letter

```
SQL> SELECT ename, job, deptno
  2  FROM emp
  3  WHERE ename LIKE '__R%';
-- (2 CONSECUTIVE underscores)
ENAME      JOB          DEPTNO
-----  -----
WARD      SALESMAN        30
MARTIN    SALESMAN        30
TURNER    SALESMAN        30
FORD      ANALYST         20
4 rows selected.
```

Arithmetic Expressions in SQL

- SQL command can contain arithmetic expressions made up of column names and constants connected by arithmetic operators (+, -, *, /)
- List employee name salary, commission, salary plus commission of employees whose commission is more than a quarter of their salary

```
SQL> SELECT ename, sal, comm, sal + comm
  2  FROM emp
  3  WHERE comm > 0.25 * sal;
    ENAME      SAL      COMM  SAL+COMM
    -----
WARD        1250       500     1750
MARTIN      1250     1400     2650
2 rows selected.
```

NULL Value Comparison in SQL

- NULL means “no value specified”, not a zero or a field with spaces
- NULL value comparison
 - Compare a variable with NULL in WHERE clause using **IS NULL** or **IS NOT NULL**
 - Don’t use **= NULL**
 - Use **ISNULL()**, **NVL()**, **IFNULL()** functions for properly handling NULL values
 - Check your RDBMS for the supported null-related functions
 - Deal with NULL values with care

NULL Value Comparison Query Examples

- List the employees in department 30 who do not receive a commission

```
SQL> SELECT ename, job  
2   FROM emp  
3  WHERE deptno = 30  
4  AND comm IS NULL;
```

ENAME	JOB
BLAKE	MANAGER
JAMES	CLERK

2 rows selected.

```
SQL> SELECT ename, job  
2   FROM emp  
3  WHERE deptno = 30  
4  AND comm = NULL;
```

no rows selected.

```
SQL> SELECT * FROM emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

Arithmetic Expressions Query Examples

- List name job, salary commission, and sum of salary and commission of employees in department 30

```
SQL> SELECT ename, job, sal, comm, sal + comm  
2   FROM emp  
3  WHERE deptno = 30;
```

ENAME	JOB	SAL	COMM	SAL+COMM
ALLEN	SALESMAN	1600	300	1900
WARD	SALESMAN	1250	500	1750
MARTIN	SALESMAN	1250	1400	2650
BLAKE	MANAGER	2850		
TURNER	SALESMAN	1500	0	1500
JAMES	CLERK	950		

6 rows selected.

```
SQL> SELECT ename, job, sal, comm, sal + NVL(comm,0)  
2   FROM emp  
3  WHERE deptno = 30;
```

ENAME	JOB	SAL	COMM	SAL+NVL(COMM,0)
ALLEN	SALESMAN	1600	300	1900
WARD	SALESMAN	1250	500	1750
MARTIN	SALESMAN	1250	1400	2650
BLAKE	MANAGER	2850		2850
TURNER	SALESMAN	1500	0	1500
JAMES	CLERK	950		950

6 rows selected.



ORDER BY Clause in SQL

- ORDER BY Clause
 - Syntax:
ORDER BY expression [ASC | DESC], ...
 - Expression
 - Column
 - Expression based on columns
 - Column alias
 - Column position in the SELECT statement (not recommended)

ORDER BY Clause in SQL (cont.)

- ORDER BY Clause
 - Sort can be ascending (ASC, the default) or descending (DESC)
 - Sort can apply to multiple columns
 - By default, nulls sort high

ORDER BY Query Examples

- List job and name of employees in department 30, order by job in ascending, and then by name in descending order

```
SQL> SELECT job, ename
  2  FROM emp
  3  WHERE deptno = 30
  4  ORDER BY job, ename DESC;
```

```
SQL> SELECT job, ename
  2  FROM emp
  3  WHERE deptno = 30
  4  ORDER BY job, 2 DESC;
```

JOB	ENAME
CLERK	JAMES
MANAGER	BLAKE
SALESMAN	WARD
SALESMAN	TURNER
SALESMAN	MARTIN
SALESMAN	ALLEN

6 rows selected.

Aggregate Functions in SQL

- Aggregate functions for a set or sets of records
 - AVG – Computes the average value
 - SUM – Computes the total value
 - MIN – Finds the minimum value
 - MAX – Finds the maximum value
 - COUNT – Counts a number of occurrences in a set

Aggregate Functions Query Examples

- Calculate the average salary for clerks

```
SQL> SELECT AVG(sal)
  2  FROM emp
  3 WHERE job = 'CLERK';

          AVG(SAL)
-----
          1037.5
```

- List the job, and the number of employees, and yearly salary of jobs where more than 2 employees are employed

```
SQL> SELECT JOB, COUNT(*), AVG(sal)*12
  2  FROM emp
  3 GROUP BY job
  4 HAVING COUNT(*) > 2;

        JOB      COUNT(*)  AVG(SAL) *12
-----
        CLERK           4       12450
        MANAGER         3       33100
        SALESMAN        4       16800
```

Johns Hopkins Engineering

Principles of Database Systems

Module 9 / Lecture 5
Structured Query Language (SQL)
The Relational DB Language I



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

UNION in SQL

- UNION (one of set operations) using SQL:
 - Union combines results of two queries. Union is a set of elements that is in one set, another set, or both sets.
 - We perform union queries when information comes from divergent sources.
 - UNION has no repeated rows.
 - UNION ALL has repeated rows.

INTERSECT in SQL

- Intersection (INTERSECT) Using SQL:
 - Intersection: what two query results have in common.
Intersection of two sets represents all elements that are members of both sets.
 - Intersection is based on exact row matches.
 - It can be simulated with “**EXISTS**”.

EXCEPT in SQL

- EXCEPT (DIFFERENCE or MINUS) Using SQL:
 - What is in the first query result is not in the second query.
 - Elements are in the original set and not in the second set.
 - The results can be simulated with “NOT EXISTS”.

SQL SET Operator Query Examples

- Two tables INSTRUCTOR and STUDENT are union compatible.
List all instructors (see Figure 6.4)

```
SQL> SELECT fname, lname FROM instructor;  
FNAME          LNAME  
-----  
John           Smith  
Ricardo        Browne  
Susan          Yao  
Francis        Johnson  
Ramesh         Shah
```

- List all students

```
SQL> SELECT fname, lname FROM student;  
FNAME          LNAME  
-----  
Susan          Yao  
Ramesh         Shah  
Johnny        Kohler  
Barbara       Jones  
Amy            Ford  
Jimmy          Wang  
Ernest         Gilbert
```

UNION and UNION ALL Query Examples

- List all names with instructors and students with UNION and UNION ALL

```
SQL> SELECT fname, lname FROM student
  2 UNION
  3 SELECT fname, lname FROM instructor;
FNAME          LNAME
-----
Amy            Ford
Barbara        Jones
Ernest          Gilbert
Francis         Johnson
Jimmy           Wang
John            Smith
Johnny          Kohler
Ramesh          Shah
Ricardo         Browne
Susan           Yao
10 rows selected.
```

```
SQL> SELECT fname, lname FROM student
  2 UNION ALL
  3 SELECT fname, lname FROM instructor;
FNAME          LNAME
-----
Amy            Ford
Barbara        Jones
Ernest          Gilbert
Francis         Johnson
Jimmy           Wang
John            Smith
Johnny          Kohler
Ramesh          Shah
Ricardo         Browne
Susan           Yao
Ramesh          Shah
Susan           Yao
12 rows selected.
```

INTERSECT (MINUS) Query Examples

- List all names that are both students and instructors

```
SQL> SELECT fname, lname FROM student  
2  INTERSECT  
3  SELECT fname, lname FROM instructor;
```

FNAME	LNAME
<hr/>	
Ramesh	Shah
Susan	Yao

Note: EXCEPT is ISO standard;
MINUS is Oracle implementation.

- List all names that are students who are not instructors

```
SQL> SELECT fname, lname FROM STUDENT  
2  MINUS  
3  SELECT fname, lname FROM INSTRUCTOR;
```

FNAME	LNAME
<hr/>	
Amy	Ford
Barbara	Jones
Ernest	Gilbert
Jimmy	Wang
Johnny	Kohler

INTERSECT (MINUS) Query Examples (cont.)

- List all names that are instructors who are not students

```
SQL> SELECT fname, lname FROM instructor  
2 MINUS  
3 SELECT fname, lname FROM student;
```

FNAME	LNAME
<hr/>	
Francis	Johnson
John	Smith
Ricardo	Browne

Note: **EXCEPT** is ISO standard; **MINUS** is Oracle implementation.

Column Prefix in SQL

- Column Prefix:
 - Used to identify the table to which a column belongs
Syntax: table.column
Example: emp.ssn; dept.dept_name
 - Required when two tables have an identical column name (e.g., PK and FK) and one of the columns is referenced in one SQL statement

Example:

```
SQL> SELECT emp.name, dept.deptno, dept.name
2   FROM emp, dept;
```

Alias for Table and Column in SQL

- Alias:
 - Used to rename an object within the SQL
 - Two types:
 - Column alias – rename long or cryptic column names
 - Table alias – rename long table names
 - Each column or table may be followed by an alias
 - Syntax:

SELECT col_name1 **[AS]** col_alias1, col_name1 **[AS]** col_alias1 ...

FROM table1 table_alias1, table1 table_alias1...;

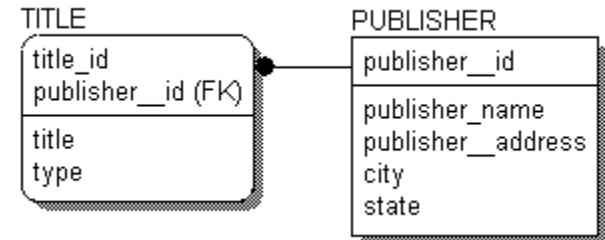
Subquery in SQL

■ Subquery:

- a SELECT statement embedded within another SELECT statement
- **Noncorrelated subquery:** evaluate from the inside out.
The outer query takes an action based on the results of the inner query.

Example: Retrieve publishers who publish the 'education' type titles

```
SELECT publisher_name  
FROM publisher  
WHERE publisher_id IN (SELECT UNIQUE publisher_id  
                      FROM title  
                      WHERE type = 'education');
```



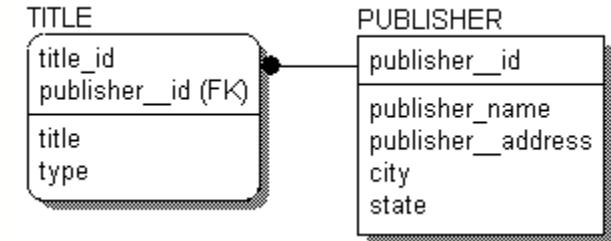
Subquery in SQL (cont.)

■ Subquery (cont.):

- **Correlated subquery:** The outer query provides the values for the inner subquery to use in its evaluation.

Example: Retrieve publishers who publish the 'education' type titles

```
SELECT publisher_name  
FROM publisher p  
WHERE EXISTS (SELECT *  
              FROM title  
              WHERE title.publisher_id = p.publisher_id AND  
                    type = 'education');
```



- EXISTS for existence is the connector for most correlated subqueries.

Subquery in SQL (cont.)

- **Subquery (cont.):**
 - Three main types of subquery connections:
 - May return a single value
 - Use a single comparison operator (<, <=, <>, >, >=)
 - May return zero or any number of items
 - Use IN, NOT IN, or with a comparison operator with ANY or ALL
 - E.g., > ANY; > ALL
 - May test existence or nonexistence
 - EXISTS, NOT EXISTS

Subquery Examples

■ Subquery (cont.):

- Subquery returns only one value.
- List all employees whose job is the same as JONES

```
SQL> SELECT ename, job  
  2  FROM emp  
  3  WHERE job =  
  4          (SELECT job  
  5            FROM emp  
  6            WHERE ename = 'JONES') ;
```

ENAME	JOB
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER

3 rows selected.

Subquery Examples (cont.)

■ Subquery (cont.):

- Subquery returns a set of values.
- List employees who earn more money than any single employee in department 30;

```
SQL> SELECT DISTINCT sal, job, ename, deptno
  2  FROM emp
  3 WHERE sal > ANY
  4       (SELECT sal
  5        FROM emp
  6        WHERE deptno = 30)
  7 ORDER BY sal DESC;
```

SAL	JOB	ENAME	DEPTNO
5000	PRESIDENT	KING	10
3000	ANALYST	FORD	20
3000	ANALYST	SCOTT	20
2975	MANAGER	JONES	20
2850	MANAGER	BLAKE	30
2450	MANAGER	CLARK	10
1600	SALESMAN	ALLEN	30
1500	SALESMAN	TURNER	30
1300	CLERK	MILLER	10
1250	SALESMAN	MARTIN	30
1250	SALESMAN	WARD	30
1100	CLERK	ADAMS	20

12 rows selected.

Subquery Examples (cont.)

■ Subquery (cont.):

- Subquery returns more than one column.
- List the employees whose job and salary are identical to that of FORD;

```
SQL> SELECT ename, job, sal
  2  FROM emp
  3  WHERE (job, sal) =
  4      (SELECT job, sal
  5       FROM emp
  6       WHERE ename = 'FORD') ;
```

ENAME	JOB	SAL
SCOTT	ANALYST	3000
FORD	ANALYST	3000

2 rows selected.

Subquery Examples (cont.)

- **Subquery (cont.):**
 - Compound query has multiple subqueries.
 - List the name, job, department number and salary of employees whose job is the same as JONES' or whose salary is at least as much as FORD's

```
SQL> SELECT ename, job, deptno, sal
  2  FROM emp
  3  WHERE job =
  4    (SELECT job
  5     FROM emp
  6     WHERE ename = 'JONES')
  7  OR sal > =
  8    (SELECT sal
  9     FROM emp
 10    WHERE ename = 'FORD')
 11 ORDER BY job, sal;
```

ENAME	JOB	DEPTNO	SAL
<hr/>			
SCOTT	ANALYST	20	3000
FORD	ANALYST	20	3000
CLARK	MANAGER	10	2450
BLAKE	MANAGER	30	2850
JONES	MANAGER	20	2975
KING	PRESIDENT	10	5000

6 rows selected.

Subquery Examples (cont.)

■ Subquery (cont.):

- List select name and job of employee in department 10 whose job is the same as any employee in department sales

```
SQL> SELECT ename, job
  2  FROM emp
  3  WHERE deptno = 10
  4      AND job IN
  5          (SELECT job
  6           FROM emp
  7           WHERE deptno IN
  8               (SELECT deptno
  9                FROM dept
 10               WHERE dname = 'SALES'));
```

ENAME	JOB
MILLER	CLERK
CLARK	MANAGER

2 rows selected.

Subquery Examples (cont.)

■ Subquery (cont.):

- List the employees who work in CHIGAGO and who have the same job as ALLEN

```
SQL> SELECT ename, loc, sal, job
  2  FROM emp, dept
  3  WHERE loc = 'CHICAGO'
  4  AND emp.deptno = dept.deptno
  5  AND job IN
  6    (SELECT job FROM emp
  7     WHERE ename = 'ALLEN')
  8  ORDER BY ename;
```

ENAME	LOC	SAL	JOB

-			
ALLEN	CHICAGO	1600	SALESMAN
MARTIN	CHICAGO	1250	SALESMAN
TURNER	CHICAGO	1500	SALESMAN
WARD	CHICAGO	1250	SALESMAN

4 rows selected.

Subquery Examples (cont.)

■ Subquery (cont.):

- Correlated query
- List the department number employee name and salary of all employees whose salary is more than the average salary of the department they work in

```
SQL> SELECT deptno, ename, sal
  2  FROM emp X
  3  WHERE sal >
  4    (SELECT AVG(sal)
  5     FROM emp
  6     WHERE X.deptno = deptno)
  7  ORDER BY deptno;
```

DEPTNO	ENAME	SAL
10	KING	5000
20	JONES	2975
20	SCOTT	3000
20	FORD	3000
30	ALLEN	1600
30	BLAKE	2850

6 rows selected.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 1 - 6

SQL - The Relational DB Language II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Join in SQL

- JOIN
 - Combine rows from two or more relations in a database
 - Conceptually easier and more obvious for the programmer than a correlated subquery
 - Exploit obvious relationships and can be used to discover new relationships

Join in SQL (cont.)

- JOIN (cont.)
 - Can join on any columns in tables
 - if joined columns match data types
 - if join operation makes sense
 - the joined columns don't need to be key attributes
 - Joined columns are key attributes in general (e.g., PK and FK)
 - Consume system resources (memory and CPU time)

Join in SQL (cont.)

■ JOIN (cont.)

- In COMPANY database, for every project located in 'Stafford', list the project number, the controlling department, and the department manager's last name, address, and birth date

Traditional JOIN

```
SELECT Pnumber, Dnum, Lname, Address, Bdate  
FROM PROJECT, DEPARTMENT, EMPLOYEE  
WHERE Dnum = Dnumber AND  
      Mgr_ssn = Ssn AND  
      Plocation = 'Stafford' ;
```

ANSI JOIN ON

```
SELECT Pnumber, Dnum, Lname, Address, Bdate  
FROM PROJECT JOIN DEPARTMENT ON Dnum = Dnumber  
      JOIN EMPLOYEE ON Mgr_ssn = Ssn  
WHERE Plocation = 'Stafford' ;
```

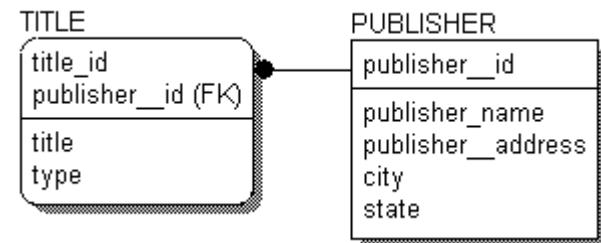
Join vs. Subquery

■ Join vs. Subquery

- May use both joins and subqueries to query multiple tables. In general, they can be used interchangeably to solve a given problem.

Example: Retrieve publishers who publish the 'education' type titles

```
SELECT publisher_name
FROM publisher p, title t
WHERE p.publisher_id = t.publisher_id AND
      t.type = 'education';
```



Join vs. Subquery (cont.)

- Join vs. Subquery (cont.)
 - Subquery can calculate an aggregate value (e.g., max, min, avg, sum, and count) on the fly and feed it back to the outer query for comparison.
 - Subquery is a good choice when you need to compare aggregates to other values.

Example: Retrieve the book title(s) with a *lowest price* in the catalog

```
SELECT title, price  
FROM catalog  
WHERE price = SELECT MIN(price) FROM catalog;
```

Join vs. Subquery (cont.)

- Join vs. Subquery (cont.)
 - Join operation provides additional options to let you edit the results from two joined tables. Join is a good choice when you want to display results from multiple tables.

Example: Retrieve publishers and authors who live in the same city

1. What is the relationship between publisher and author tables?
2. How to handle a many-to-many relationship?

```
SELECT publisher_name, author_name  
FROM publisher P, author A  
WHERE P.city = A.city;
```

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 2

SQL - The Relational DB Language II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Cross Join

■ Cross Join

- CROSS JOIN returns the Cartesian product of two tables.

Traditional Way:

```
SELECT ColumnList FROM table1, table2
```

Note: A query for getting information from two tables without a join condition may be a programming mistake.

SQL Standard Way:

```
SELECT [DISTINCT | ALL] { * | ColumnList }
FROM table1 CROSS JOIN table2
```

```
SQL> SELECT * FROM emp, dept;
SQL> SELECT * FROM emp CROSS JOIN dept;
. .
56 rows selected. (14 employees and 4 departments)
```

Inner Join

■ Inner Join

- **INNER JOIN** with **ON**
returns only the rows that
meet the join condition
indicated in the ON clause

```
SELECT [DISTINCT | ALL]
{ * | ColumnList }
FROM table1 [INNER] JOIN table2 ON
    table1.c1 = table2.c1;
```

```
SQL> SELECT empno, ename, emp.deptno, dname, loc
2  FROM emp JOIN dept ON emp.deptno = dept.deptno;
```

EMPNO	ENAME	DEPTNO	DNAME	LOC
7369	SMITH	20	RESEARCH	DALLAS
7499	ALLEN	30	SALES	CHICAGO
7521	WARD	30	SALES	CHICAGO
7566	JONES	20	RESEARCH	DALLAS
7654	MARTIN	30	SALES	CHICAGO
7698	BLAKE	30	SALES	CHICAGO
7782	CLARK	10	ACCOUNTING	NEW YORK
7788	SCOTT	20	RESEARCH	DALLAS
7839	KING	10	ACCOUNTING	NEW YORK
7844	TURNER	30	SALES	CHICAGO
7876	ADAMS	20	RESEARCH	DALLAS
7900	JAMES	30	SALES	CHICAGO
7902	FORD	20	RESEARCH	DALLAS
7934	MILLER	10	ACCOUNTING	NEW YORK

Inner Join (cont.)

■ Inner Join (cont.)

- Traditional **JOIN** returns only the rows that meet the join condition in the **WHERE** clause

```
SELECT [DISTINCT|ALL]
{ * | ColumnList }
FROM table1, table2
WHERE table1.c1 = table2.c1;
```

```
SQL> SELECT empno, ename, emp.deptno, dname, loc
  2 FROM emp, dept
  3 WHERE emp.deptno = dept.deptno;
```

EMPNO	ENAME	DEPTNO	DNAME	LOC
7369	SMITH	20	RESEARCH	DALLAS
7499	ALLEN	30	SALES	CHICAGO
7521	WARD	30	SALES	CHICAGO
7566	JONES	20	RESEARCH	DALLAS
7654	MARTIN	30	SALES	CHICAGO
7698	BLAKE	30	SALES	CHICAGO
7782	CLARK	10	ACCOUNTING	NEW YORK
7788	SCOTT	20	RESEARCH	DALLAS
7839	KING	10	ACCOUNTING	NEW YORK
7844	TURNER	30	SALES	CHICAGO
7876	ADAMS	20	RESEARCH	DALLAS
7900	JAMES	30	SALES	CHICAGO
7902	FORD	20	RESEARCH	DALLAS
7934	MILLER	10	ACCOUNTING	NEW YORK

Inner Join (cont.)

■ Inner Join (cont.)

- JOIN with USING returns only the rows with matching values in the *common* columns indicated in the USING clause

```
SELECT [DISTINCT|ALL]
{ * | ColumnList }
FROM table1 JOIN table2 USING (c1);
```

```
SQL> SELECT empno, ename, emp.deptno, dname, loc
2  FROM emp JOIN dept USING(deptno);
```

EMPNO	ENAME	DEPTNO	DNAME	LOC
7369	SMITH	20	RESEARCH	DALLAS
7499	ALLEN	30	SALES	CHICAGO
7521	WARD	30	SALES	CHICAGO
7566	JONES	20	RESEARCH	DALLAS
7654	MARTIN	30	SALES	CHICAGO
7698	BLAKE	30	SALES	CHICAGO
7782	CLARK	10	ACCOUNTING	NEW YORK
7788	SCOTT	20	RESEARCH	DALLAS
7839	KING	10	ACCOUNTING	NEW YORK
7844	TURNER	30	SALES	CHICAGO
7876	ADAMS	20	RESEARCH	DALLAS
7900	JAMES	30	SALES	CHICAGO
7902	FORD	20	RESEARCH	DALLAS
7934	MILLER	10	ACCOUNTING	NEW YORK

Natural Join

- Natural Join
 - **NATURAL JOIN** returns only the rows with matching values in the matching columns. The matching columns must have the same names and similar data types

```
SELECT [DISTINCT | ALL] { * | ColumnList }
      FROM table1 NATURAL JOIN table2
```
 - The NATURAL and USING keywords are mutually exclusive. They cannot be used together.

Natural Join (cont.)

- Natural Join (cont.)
 - With NATURAL JOIN only:

```
SQL> SELECT empno, ename, deptno, dname, loc
  2  FROM emp NATURAL JOIN dept
  3 WHERE deptno = 30;
```

EMPNO	ENAME	DEPTNO	DNAME	LOC
7499	ALLEN	30	SALES	CHICAGO
7521	WARD	30	SALES	CHICAGO
7654	MARTIN	30	SALES	CHICAGO
7698	BLAKE	30	SALES	CHICAGO
7844	TURNER	30	SALES	CHICAGO
7900	JAMES	30	SALES	CHICAGO

6 rows selected.

- Natural Join (cont.)
 - With NATURAL JOIN and WHERE clause:

```
SQL> SELECT *
  2  FROM emp NATURAL JOIN dept
  3 WHERE emp.deptno = 30;
SELECT * FROM emp NATURAL JOIN dept WHERE emp.deptno = 30
*
ERROR at line 1:
ORA-25155: column used in NATURAL join cannot have qualifier
```

Outer Join

- Outer Join
 - **OUTER JOIN** returns not only the rows matching the join condition, but also the rows with unmatched values.
 - **LEFT OUTER JOIN** returns rows with matching values and the rows in table1 without matching values.

```
SELECT *
FROM table1 LEFT [OUTER] JOIN table2 ON
table1.c1 = table2.c1;
```
 - **RIGHT OUTER JOIN** returns rows with matching values and the rows in table2 without matching values.

```
SELECT *
FROM table1 RIGHT [OUTER] JOIN table2 ON
table1.c1 = table2.c1;
```

Outer Join (cont.)

■ Outer Join (cont.)

- **FULL OUTER JOIN** returns rows with matching values and the rows in table1 as well as table2 without matching values.

```
SELECT *
FROM table1 FULL [OUTER]
JOIN table2 ON
    table1.c1 = table2.c1;
```

```
SQL> SELECT *
  2  FROM emp LEFT OUTER JOIN dept ON (emp.deptno = dept.deptno);
...
14 rows selected.

SQL> SELECT *
  2  FROM emp RIGHT OUTER JOIN dept ON (emp.deptno = dept.deptno);
...
15 rows selected.

SQL> SELECT *
  2  FROM emp FULL OUTER JOIN dept ON (emp.deptno = dept.deptno);
...
15 rows selected.
```

Self Join

- **SELF JOIN** has a table that joins itself, a recursive relationship.
 - List all employees and their supervisors.

```
SQL> SELECT emp.empno, emp.ename, mgr.empno AS manger_empno, mgr.ename AS mgr_name
  2  FROM emp JOIN emp mgr ON (emp.mgr = mgr.empno);
   EMPNO ENAME      MANGER_EMPNO MGR_NAME
----- -----
    7369 SMITH          7902 FORD
    7499 ALLEN          7698 BLAKE
    7521 WARD           7698 BLAKE
    7566 JONES          7839 KING
    7654 MARTIN         7698 BLAKE
    7698 BLAKE          7839 KING
    7782 CLARK           7839 KING
    7788 SCOTT           7566 JONES
    7844 TURNER          7698 BLAKE
    7876 ADAMS            7788 SCOTT
    7900 JAMES           7698 BLAKE
    7902 FORD             7566 JONES
    7934 MILLER          7782 CLARK
13 rows selected.
```

Recursive Closure Operation

■ Recursive Closure Operation

- For a recursive relationship, a query to get a hierarchical relation among records.
- This is a hierarchical query and SQL standard which uses recursive common table expressions: <http://msdn.microsoft.com/en-us/library/ms186243.aspx>

Example: Retrieve all employees that work under “JONES” using Oracle:

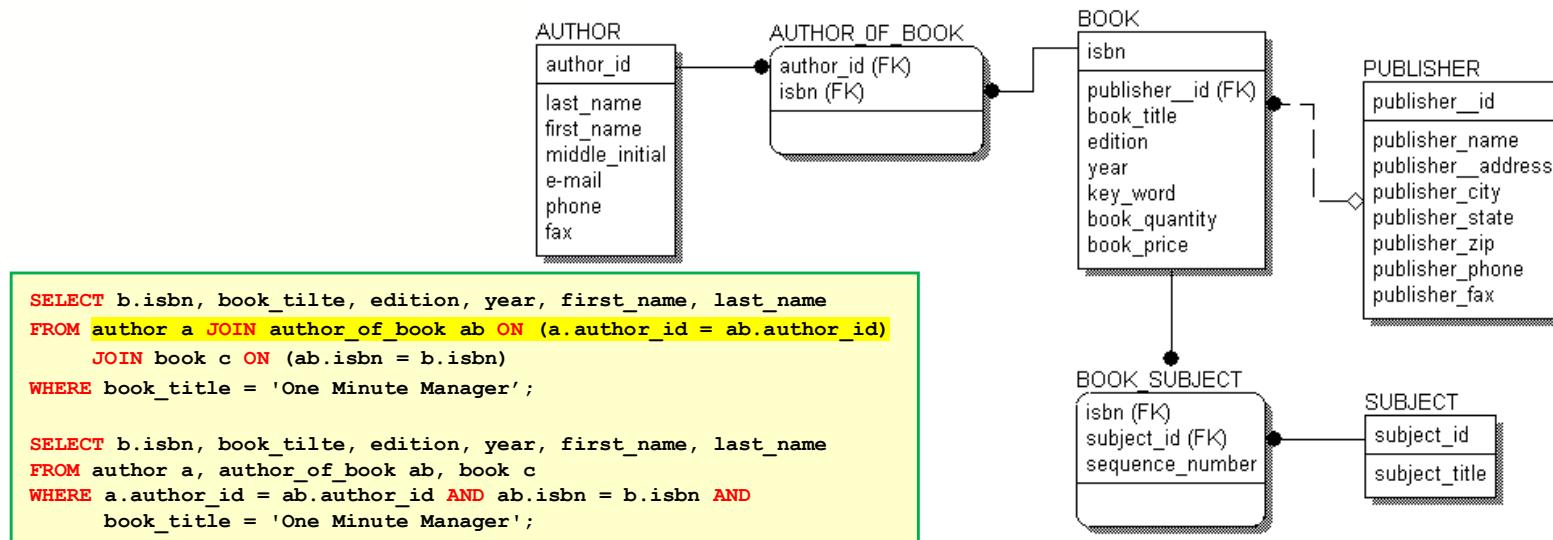
```
SQL> SELECT LEVEL, LPAD (' ', 2 * (LEVEL - 1)) || ename "employee", empno, mgr
  FROM emp START WITH ename = 'JONES'
  CONNECT BY PRIOR empno = mgr;
```

LEVEL	employee	EMPNO	MGR
1	JONES	7566	7839
2	SCOTT	7788	7566
3	ADAMS	7876	7788
2	FORD	7902	7566
3	SMITH	7369	7902

Join for a Many-to-Many relationship

- Join three tables for a M:N relationship

BOOK STORE or LIBRARY ERD (A Subset)



Q1: Find out an author(s) for a book titled 'One Minute Manager' and the book's related information

Q2: Find out an author(s) for all books order by title name and the book's related information

Q3: Find out books by a specific author 'John Smith' and the book's related information

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 3

SQL - The Relational DB Language II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

CASE Expression

■ CASE Expression

```
CASE ColumnName  
    WHEN condition1 THEN result1  
    WHEN condition2 THEN result2  
    ...  
    ELSE result  
END
```

```
SELECT fname, lname, dept_id,  
    (CASE title  
        WHEN LIKE 'Senior%' THEN '10%'  
        WHEN LIKE 'Junior%' THEN '5%'  
        ELSE '0'%  
    END) AS bonus  
FROM employee;
```

```
SELECT fname, lname,  
    (CASE dept_id  
        WHEN 1 THEN 'Accounting'  
        WHEN 2 THEN 'Finance'  
        WHEN 3 THEN 'Engineering'  
        ELSE 'Not required'  
    END) AS department  
FROM employee;
```

Data Manipulation - Insertion

■ Data Manipulation - Insertion

- Add a single row to a table

```
INSERT INTO TableName [ColumnList] VALUES (valuelist);
```

```
INSERT INTO emp  
VALUES (7954, 'CARTER', 'CLERK', 7698, '7-APR-82', 1000, NULL, 30);
```

```
INSERT INTO emp(empno, ename, job, mgr)  
VALUES (7980, 'SMITH', 'MANAGER', 3839);
```

- Add multiple rows from one table to another table

```
INSERT INTO TableName [ColumnList]
```

```
SELECT ...;
```

```
INSERT INTO bonus (ename, job, sal, comm)  
SELECT ename, job, sal, comm  
FROM emp  
WHERE job = 'MANAGER' OR  
comm > 0.25 * sal;
```

- Add/copy some or all columns from one table into a **new** table

```
SELECT {*|ColumnList} INTO NewTable FROM OldTable WHERE condition;
```

Data Manipulation - Modification

■ Data Manipulation - Modification

- Change the contents of existing rows

```
UPDATE TableName  
SET ColumnName1 = value1 [, ...]  
WHERE ...
```

- Update a field in one row of a table

```
UPDATE emp  
SET sal = 3300  
WHERE empno = 7788;
```

- Update multiple fields of multiple rows in a table

```
UPDATE emp  
SET sal = sal * 1.15, comm = comm + 100  
WHERE 'SALESMAN' AND deptno = 30;
```

MERGE

- MERGE (also called upsert) statements to INSERT new records or UPDATE existing records depending on whether condition matches.

```
MERGE INTO tablename USING table_reference ON (condition)
  WHEN MATCHED THEN
    UPDATE SET column1 = value1 [, column2 = value2 ...]
  WHEN NOT MATCHED THEN
    INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...]);
```

Tablename is the **target** and **table_reference** is the **source** (table/view/subquery).
Check your RDBMS manual to see whether it supports standard MERGE syntax.

Data Manipulation – Deletion

■ Data Manipulation - Deletion

- Delete rows from a table

```
DELETE FROM TableName  
WHERE ...
```

- Delete one row of a table

```
DELETE FROM emp  
WHERE ename = 'WARD';
```

- Delete multiple rows in a table with or without a condition

```
DELETE FROM bonus; -- delete all rows in the bonus table  
  
DELETE FROM bonus  
WHERE job IN  
      (SELECT job  
       FROM emp  
       WHERE empno = 7566);
```

Data Manipulation – Deletion (cont.)

- Delete rows from a table using TRUNCATE TABLE.
 - Is a Data Definition Language (DDL) operation
 - Quickly removes all data from a table, typically bypassing a number of integrity enforcing mechanisms
 - Needs to check RDBMS specific implementations

TRUNCATE TABLE TableName

Note: Use TRUNCATE TABLE statement carefully; particularly for a production system. This is a best practice that one may consider doing a backup on a production system before executing TRUNCATE TABLE or DROP TABLE DDLs.

Views in SQL

- A view is a virtual or derived table. The rows of a view do not exist until they are derived from base tables at run time.
- Changes to any of the base tables in the defining query are immediately reflected in the view.
- A view presents current and dynamic information to users regardless of the constantly changing underlying source tables.

Views in SQL (cont.)

- Views can be used for:
 - Restricting access (additional level of table security)
 - Providing referential integrity
 - Presenting tables to users in various forms
 - Pre-joining base tables for easily developing an application
 - Pre-packaging complex queries

Views in SQL (cont.)

- Create a view by embedding a subquery for a subset of columns and/or rows, column expressions from one or more tables or views:

CREATE VIEW ViewName AS subquery

Example: Create a view with employee last name, firstname, and department name

```
CREATE VIEW v_emp_dept
AS SELECT emp.lname, emp.fname, dept.d_name
      FROM emp, dept
     WHERE emp.dept_id = dept.dept_id
```

Views in SQL (cont.)

- Views in practice:
 - Creating a join view with joined tables (e.g., AUTHOR, AUTHER_OF_BOOK, and BOOK) to reduce query complexity
 - Creating views for security and role-based access controls
 - An EMP view excluding sensitive employee attributes
 - Creating views to support multi-user database for customization
 - Views for marketing, sales, customer, administration roles

Views in SQL (cont.)

- Updating a view to a base table may be simple. However, not all views can be updated:
 - Expressions
 - Aggregate functions
 - References to views that are not updatable
 - GROUP BY or HAVING clauses
 - Set operations with multiple tables

View Materialization

- View Materialization – Materialized View
 - View may cause performance issues. View materialization stores the view as a temporary table.
 - Oracle uses a materialized view (used to called a snapshot) containing the results of a query.
 - A materialized view can be used to store copies of remote data on a local system for replication purposes.
 - A materialized aggregated view or joined view can be used for data warehouse purposes as a fact table.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 2

SQL - The Relational DB Language II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Indexing to Improve Query Performance

- Create indexes on a column(s) on database tables. Indexes are an explicitly created schema object. They are stored independently.

Syntax:

```
CREATE INDEX index_name  
ON table_name(column_name)
```

Example:

```
CREATE INDEX emp_lname ON emp(lname);
```

Note: These statements have been *removed* from SQL2 because they specify physical access paths - not conceptual or logical concepts.

Indexing to Improve Query Performance (cont.)

- Create unique indexes for rows with unique values in the indexed column(s). A column with unique value normally is important for query and it is good to create a unique index.

Syntax:

```
CREATE UNIQUE INDEX index_name  
ON table_name(column_name)
```

Example:

```
CREATE UNIQUE INDEX emp_ssn ON emp(ssn);
```

Indexing to Improve Query Performance (cont.)

- Delete or drop an index(es):

Syntax:

DROP INDEX index_name

Example:

DROP INDEX emp_lname;

Dynamic SQL

- Refers to DDL, DML, and query statements that are constructed, parsed, and executed at runtime:
 - May need input from the users such as the columns they want to see or some elements of the WHERE clause

SQL Injection

- SQL injection is a type of security attack against databases in which the attacker enters SQL code to a Web form input box to maliciously gain access to resources or make changes to data.

Example:

Login: ' OR '' = '

Password: Anything' OR '' = ''

```
SELECT username FROM Customer  
WHERE username = '' OR '' = ''  
AND password = 'Anything' OR '' = '';
```

SQL Injection (cont.)

- Let's treat **emp** table storing user **login** table. You can treat the **username** column as **ename** column; and **job** column as **password** column.

Login: abc

Password: xyz

```
SELECT * FROM emp
WHERE ename = 'abc'    -- ename is username and job is password
  AND job = 'xyz';
No rows selected
```

Login: 'OR ''='

Password: Anything' OR '' = '

```
SELECT * FROM emp
WHERE ename = ''OR '' = ''
  AND job = 'Anything' OR '' = ''';
...
14 rows selected
```

Note: Unexpected input with unexpected output!

SQL Injection (cont.)

- Web applications that use dynamic content without data validations are vulnerable to SQL injection.
 - Illegal access to databases
 - Steal sensitive information
 - Maliciously insert, modify or delete information
- Automated SQL injection programs are now available, and as a result, both the likelihood and the potential damage of an exploit has increased enormously.

SQL Injection (cont.)

- The fundamental issues
 - Programmers may not know secure coding practice.
 - Security is not sufficiently emphasized in software development.
 - Developers focus on the legal values of parameters and how they should be utilized without considering invalid input values.
 - The testers of web applications don't do in-depth testing to discover invalidated input values and their corresponding unexpected behaviors.

SQL Injection (cont.)

- How to protect the integrity of web sites and applications:
 - Controlling the types and numbers of characters accepted by input boxes (data sanitization).
 - Calling stored procedure to login.
 - Access control over sensitive tables and columns.
- How to identify application vulnerabilities:
 - Using “AppScan” and “Netsparker” to discover vulnerable web pages.
 - Using “Fortify Software” and “Checkmarx” to scan source code.
 - Using “Core Impact” by Core Security, a penetration testing tool to validate discovered vulnerabilities.
 - Manually validating vulnerability findings if necessary.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 5
SQL - The Relational DB Language II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

SQL Programming Language

- Advanced users, such as developers or DBAs, can use vendors' tools to communicate with DBMSs and support complex business processes.
- SQL*Plus, is a tool as a command-line interface with Oracle database (**sql>**).
- Common tools for other DBMSs are:
 - "isql" in Sybase and SQL Server
 - "db2" in IBM DB2
 - "psql" in PostgreSQL
 - "mysql" in MySQL
- SQL lacks programming features such as variable, constant declarations, flow controls, exception handling, and modulation.

SQL Programming Language (cont.)

- The server processes SQL statements one at a time. Each SQL statement results in a separate call from the client to the server, causing a overhead, especially across web or a network.
- A high-level language such as Pro*C/C++, Java can also embed SQL and perform database operations.

SQL/Persistent Stored Modules (PSM)

- SQL/PSM is an ANSI SQL extension with procedural programmability.
- Most commercial DBMSs implemented this feature before the standard.
- Procedural Language/SQL (PL/SQL) is Oracle's procedural extension to SQL.
 - Embedding SQL in a Programming Language (PL/SQL) to manipulate complex database operations.
- Similarly, Sybase and Microsoft has Transact-SQL; IBM has SQL PL (Procedural Language)

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation:
 - PL/SQL supports modular programming, declare identifiers, and uses them in a program with procedural language control structures, and error handling.
 - PL/SQL is built in a block construct.

```
[DECLARE TYPE / item / FUNCTION / PROCEDURE declarations] -- optional
BEGIN -- mandatory
  statements
  [EXCEPTION -- optional
    exception handlers]
END; -- mandatory
```

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation (cont.)
 - PL/SQL language can be written using the basic control structures such as SEQUENCE, SELECTION and ITERATION.
 - Conditional IF statement:

```
IF (condition) THEN <sql statements>
[ELSIF (condition) THEN <sql statements>]
[ELSE <sql statements>]
END IF;
```

- Conditional CASE statement:

```
CASE (operand)
[WHEN (operand list) | WHEN (search condition)
    THEN <sql statements>]
[ELSE <sql statements>]
END CASE;
```

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation (cont.)
 - Iteration LOOP, WHILE, or FOR statements:

```
WHILE (condition)
    LOOP <sql statements>
END LOOP;

FOR iVariable IN lowerbound..upperbound
LOOP
    <sql statements>
END LOOP;
```

- The set of rows returned by a multi-row query is called the active set. An explicit **cursor** is equivalent to a *point* to the current row in the active set. This allows a program to process the rows one at a time.

SQL/Persistent Stored Modules (cont.)

- Example: Oracle PL/SQL Implementation (cont.)
 - Embedding SQL in a programming language
 - Use dot notation to reference individual fields

Example: Calculate bonus based on department

```
DECLARE
    CURSOR c1 IS
        SELECT ename, salary, dept_id FROM emp;
        . . .
BEGIN
    . . .
    FOR emp_rec IN c1 LOOP
        . . .
        salary_sum := salary_sum + emp_rec.sarlary;
        . . .
    END LOOP;
END;
```

SQL/Persistent Stored Modules (cont.)

■ General Constraints

- Data manipulation to tables may have complex rules beyond common structural constraints such as PKs, FKs, NOT NULL, Unique, Domain, and Check

Example: An employee cannot register two cars for parking permits.

```
CREATE ASSERTION TooManyCars_Constraint
    CHECK (NOT EXIST (SELECT emp_id
                      FROM CAR
                      GROUP BY emp_id
                      HAVING COUNT(*) > 2));
```

SQL/Persistent Stored Modules (cont.)

- General Constraints (cont.)
 - Business processes need general constraints to support complex rules.
 - Register a course with prerequisite courses and a minimum GPA requirement.
 - Withdraw cash less than the account balance or more than a daily allowed maximum.
 - Assign no more than 10 tasks to an employee or no more than 5 tasks to an employee who has three high-priority tasks.
 - Create an audit trail of all rows inserted into the Customer_Order table.
 - Triggers can be used for complex constraints. Not all DBMSs support triggers. If not, applications need to implement general constraints.

Triggers

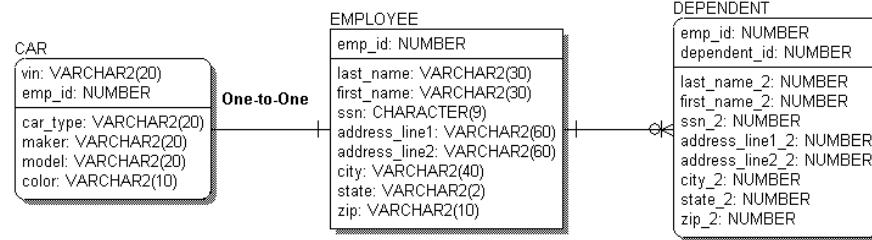
- Triggers:
 - Programs that are executed automatically in response to a change in the database for enforcing constraints or business rules. Trigger can be configured to fire or execute, either before or after the trigger event.
 - Event-Condition-Action:

```
<trigger> ::= CREATE TRIGGER <trigger name>
              ( BEFORE | AFTER ) <triggering events>
              ON <table name>
              [ FOR EACH ROW ]
              [ WHEN <condition> ]
              <trigger actions> ; -- trigger body

<triggering events> ::= <trigger event>
                         {OR <trigger event> }
<trigger event> ::= INSERT | DELETE | UPDATE [ OF <column name> {, <column name>} ]
<trigger action> ::= <PL/SQL block>
```

Triggers (cont.)

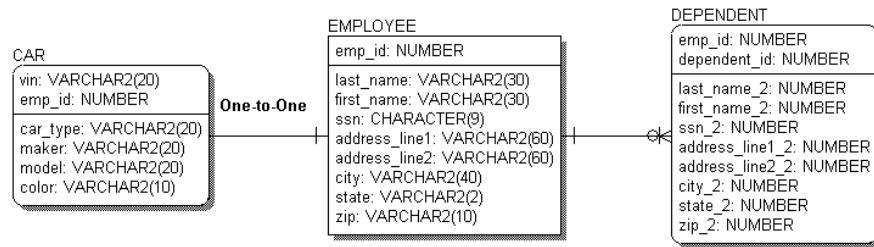
- Triggers (cont.)
 - Example: An Update Trigger



```
-- CREATE TRIGGER using Oracle Syntax:  
CREATE TRIGGER tU_CAR AFTER UPDATE ON car FOR EACH ROW  
-- UPDATE trigger on CAR  
DECLARE numrows INTEGER;  
BEGIN  
    /* EMPLOYEE owns CAR on CHILD UPDATE RESTRICT */  
    SELECT count(*) INTO numrows  
    FROM employee  
    WHERE :new.emp_id = EMPLOYEE.emp_id;  
    IF ( numrows = 0 ) THEN  
        raise_application_error(-20007,  
        'Cannot UPDATE "car" because "employee" does not exist.');
```

Triggers (cont.)

- Triggers (cont.)
 - Example: An Insert Trigger



```
-- CREATE TRIGGER using Oracle Syntax:  
CREATE TRIGGER tI_CAR BEFORE INSERT ON car FOR EACH ROW  
-- INSERT trigger on CAR  
DECLARE numrows INTEGER;  
BEGIN  
    /* EMPLOYEE owns only ONE car on CHILD INSERTION */  
    SELECT count(*) INTO numrows  
    FROM car  
    WHERE :new.emp_id = EMPLOYEE.emp_id;  
    IF ( numrows = 1 ) THEN  
        raise_application_error(-20008,  
        'Cannot INSERT "car" because "employee" can only have one car.');
```

```
    END IF;  
END ;
```

Stored Procedures, Functions and Packages

- Stored Procedures, Functions and Packages:
 - Stored procedures and functions are similar to other high level programming languages to provide modulation, code reusability and maintainability.

```
CREATE OR REPLACE PROCEDURE Procedure_Name [(<parameter-list>)] AS  
BEGIN  
    <executable-section>  
    [exception  
        <exception-section>]  
END ;
```

- A parameter list can be used as inputs, outputs or both. A procedure can return a set of values through the parameter list.

Stored Procedures, Functions and Packages (cont.)

- Stored Procedures, Functions and Packages (cont.)
 - A function returns a value to the calling program.

```
CREATE OR REPLACE FUNCTION Function_Name [ (<parameter-list>) ]  
return <datatype> AS  
BEGIN  
    <executable-section>  
    [exception  
        <exception-section>]  
END ;
```

- A package is a collection of related stored procedures, functions, and variables.

Johns Hopkins Engineering

Principles of Database Systems

Module 10 / Lecture 6
SQL - The Relational DB Language II



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Object-relational Features in SQL

- Object-relational DB supports
 - Extended datatype
 - Scalar – CHAR, NCHAR (Unicode, national character set), VARCHAR2, NVARCHAR2, NUMBER, DATE, TIMESTAMP, BFILE
 - Collection – VARRAY, TABLE
 - Relationship – REF (for object table only)
 - User-defined data types
 - Objects with attributes and member methods (procedures or functions)
 - Large objects (LOB) such as BLOB, CLOB, NCLOB

Object-relational Features in SQL (cont.)

- Creating Object Types
 - An object type consists of built-in datatypes or object types.
 - Syntax in SQL:
 - **CREATE TYPE**
 - **DROP TYPE**
 - **ALTER TYPE**
 - **GRANT/REVOKE TYPE**
 - After a type is created, a constructor method is automatically created.
 - Constructor is used to create specific instances of objects.

Object-relational Features in SQL (cont.)

- Creating Relational Tables with Object Types

- Creating an object type

```
CREATE TYPE person_type AS OBJECT  
  (LastName  VARCHAR2(20),  
   FirstName  VARCHAR2(20),  
   Phone     VARCHAR2(12),  
   DOB       DATE, ...);
```

- Creating a table with an object type

```
CREATE TABLE employee  
  (emp_id    INTEGER PRIMARY KEY,  
   emp       person_type);
```

- Inserting an employee record

```
INSERT INTO employee VALUES (5, person_type('Smith', 'John', '301-420-7700', To_Date(  
  '12/17/1978', 'MM/DD/YYYY'), ...));  
INSERT INTO employee VALUES (6, person_type('Jones', 'Lynn', '410-731-4968', To_Date(  
  '06/08/1980', 'MM/DD/YYYY'), ...));
```

- Query an employee record using dot notation

```
SELECT emp_id, e.emp.FirstName, e.emp.LastName, e.emp.dob FROM employee e;
```

Object-relational Features in SQL (cont.)

■ Creating An Object Table

- An object table is a table whose rows are all objects with **object identifier** (OID) values.
- Store object instances in rows:

```
CREATE TYPE person_type AS OBJECT  
  (LastName    VARCHAR2(20),  
   FirstName   VARCHAR2(20),  
   Phone       VARCHAR2(12),  
   DOB         DATE, ...);  
  
CREATE TABLE emp_table OF person_type;  
-- Table emp_table is based on person_type datatype --
```

Object-relational Features in SQL (cont.)

- Accessing An Object in An Object Table
 - An object reference (REF) is a system generated value to locate an object in an object table.
 - A REF consists of the target's object's OID, and object table identifier, and a database identifier.
 - A REF can be used to define relationships between objects, a one-to-one unidirectional association.

Object-relational Features in SQL (cont.)

- Creating A Member Method in An Object
 - An object type can have zero or more member methods and use the `object.method` to access a member method.

```
CREATE TYPE emp_type AS OBJECT
(emp_id    NUMBER,
 LastName  VARCHAR2(20),
 FirstName   VARCHAR2(20),
 Phone      VARCHAR2(12),
 HireDate   DATE,
 MEMBER FUNCTION days_joined RETURN NUMBER, ...);

CREATE TYPE BODY emp_type
(MEMBER FUNCTION days_joined RETURN NUMBER IS
BEGIN
    RETURN floor (sysdate - hiredate);
END; );
```

Object-relational Features in SQL (cont.)

- VARRAY Type
 - A varying-length array (VARRAY) is a collection of similar items
 - The array is an ordered set of items with two attributes
 - Count: Current number of elements
 - Limit: Maximum array size
- Data in a VARRAY stored inline if size < 4Kb
Index is not supported

Object-relational Features in SQL (cont.)

- VARRAY Example
 - Use the `object.method` to access a member method

```
CREATE TYPE phone_type AS OBJECT
(Phone      VARCHAR2(12),
 Description VARCHAR2(15) );

CREATE TYPE phone_list_type AS VARRAY(10) OF phone_type;

CREATE TYPE emp_type AS OBJECT
(emp_id      NUMBER,
 LastName    VARCHAR2(20),
 FirstName   VARCHAR2(20),
 Phone_No    phone_list_type,
 HireDate    DATE, ...);

CREATE TABLE employee OF emp_type;
```

Object-relational Features in SQL (cont.)

- Nested Tables
 - A table stored within a table is called a nested table.
 - It is suitable a master-detailed relationship or one-to-many relationship.
 - Query is allowed to select nested rows.
 - Nested tables are stored *out-of-line*.
 - DMBS supports indexes for nested tables.

Object-relational Features in SQL (cont.)

- Nested Table Example
 - How to create a nested table and use it

```
CREATE TYPE phone_type AS OBJECT
(Phone          VARCHAR2(12),
 Description    VARCHAR2(15) );

CREATE TYPE phone_nested_type AS TABLE OF phone_type;

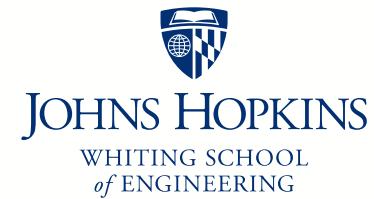
CREATE TYPE employee
(emp_id        NUMBER,
 LastName      VARCHAR2(20),
 FirstName     VARCHAR2(20),
 Phone_No      phone_nested_type,
 HireDate      DATE, ...)

NESTED TABLE Phone_No STORE AS emp_phone_nested_table_seg;
(Note: Nested table stored out-of-line in its own segment)
```

Johns Hopkins Engineering

Principles of Database Systems

Module 11 / Lecture 1
File Structures and Indexes

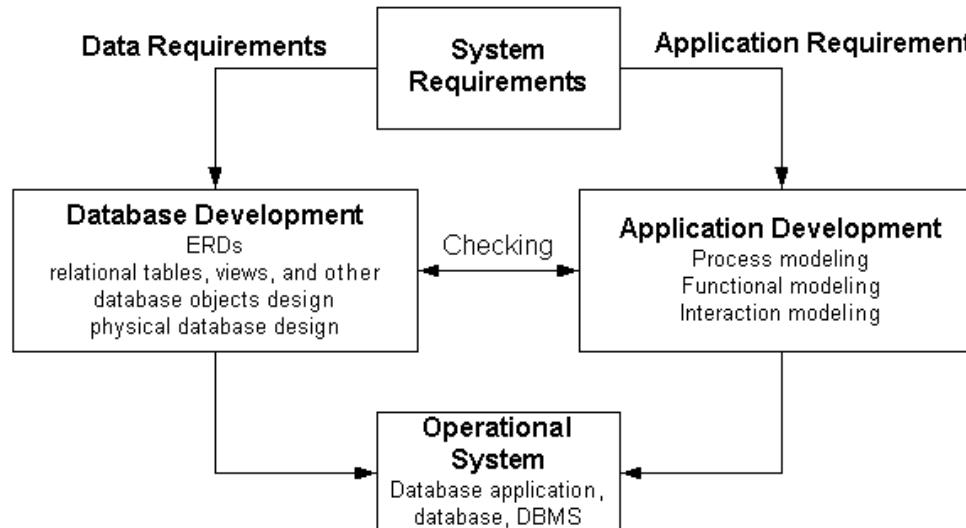


Enterprise Architecture

- Business Architecture:
 - Operating Scenarios
 - Business Process Model, Data Flow Diagram, UML Use Case
- Information Architecture:
 - Conceptual, Logical, and Physical Models, UML Class diagram, XML model
- Application Architecture:
 - Functional Model, System/Interface Model, Reuse/Collaboration Models
- Technology Architecture:
 - OS, HW, DBMS, Standards, Network, Communications, Security

Database and Database Application Development

■ Paths for Database and Database Application Development



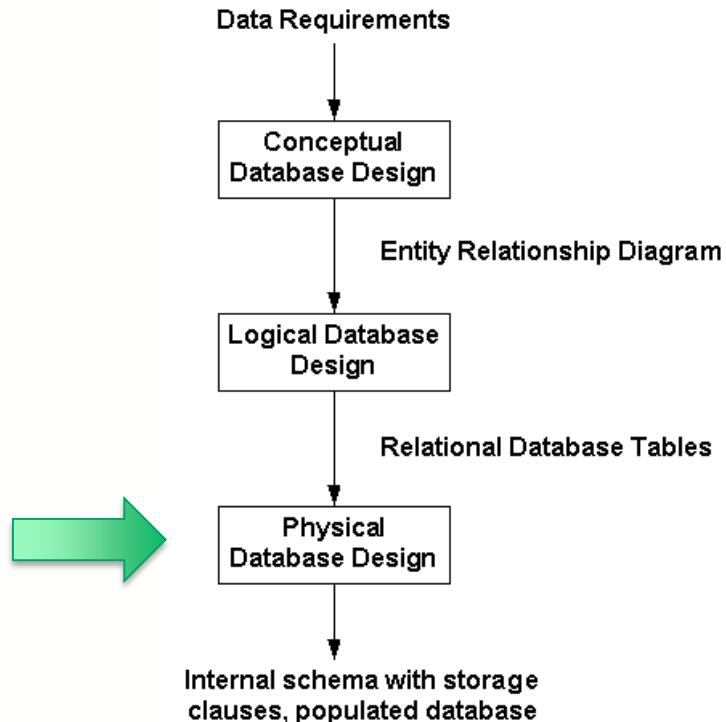
Interaction between database and application development

Business Rules

- Structural assertion
 - DDLs with constraints and domains
- Derivation
 - GPA, line total, grand total
- Action assertion
 - Decision logic – type of services/members, course registration, task assignments based on rules and roles
 - Calculation formula – sales tax

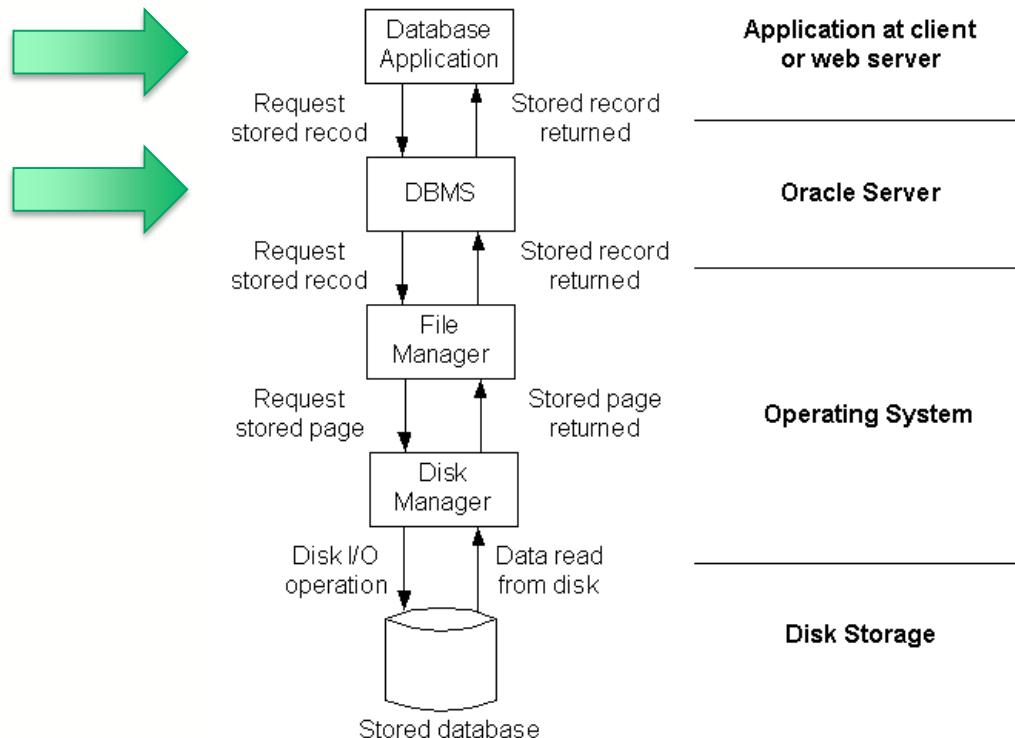
Database Design Process

■ Physical Database Design



Application, DBMS, and I/O Process

■ Data Retrieval Process Via a Database Application



Record Storage

- **Computer storage medium:**
 - Primary storage: main memory, cache memory for fast data access.
 - Second storage: magnetic disks, optical disks, tapes, and drums. They have a much larger capacity, are less expensive, but slower access than that of primary storage.
 - Flash memory (sometimes called "flash RAM") is a type of constantly-powered nonvolatile memory that can be erased and reprogrammed in units of memory called *blocks*.

Disk Devices

- The physical divisions of a disk are cylinders (for a disk pack), tracks, sectors (arc of a track), and blocks (see Figures 16.1 and 16.2).
- A disk is a random access addressable device. Data transfer between main memory and disk is in units of *blocks*.
- The process has seek time (locating the correct track by the head), rotational delay (pointing to the right block), and block transfer time in milliseconds. Locating data on a disk and transferring a disk block is in the order of **milliseconds** while CPU processing time is in the order of **microseconds** to **nanoseconds**.

Computer Hardware Improvement Rates

■ Computing component improving rate comparison

Computing components	Improving rate/year	Comments
RAM capacities	133% to 200%	10^{-9} sec.
CPU	33% to 50%	
Disk access time	Less than 10%	Major slowdown
Disk transfer rate	20%	Major slowdown
Disk capacities	50%	

Disk I/O remains the bottleneck for HW improvement

(Registers ($1\text{-}9 \times 10^{-9}$ sec.), cache ($10\text{-}40 \times 10^{-9}$ sec.), Disk ($2\text{-}3 \times 10^{-3}$ sec.))

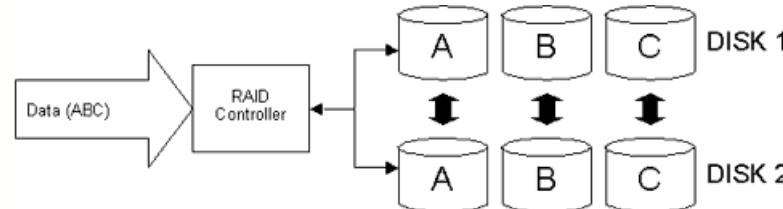
Redundant Array of Independent Disks (RAID)

- RAID is used for performance improvement and redundancy.
- Major objectives are to improve disk performance and reliability.
 - Improves performance (employs parallelism concepts to improve access time)
 - Improves reliability (uses redundancy and error checking codes)
 - Increases storage capacity since it is viewed as a large logical disk (comprised of several physical hard drives)

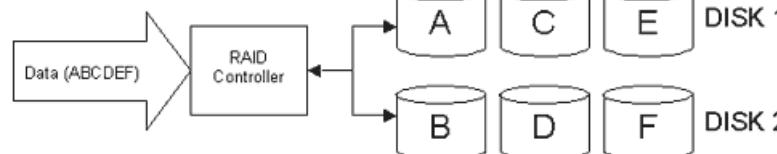
Redundant Array of Independent Disks (RAID) (cont.)

■ RAID Concepts

- Logical drive encapsulating several physical drivers
- Mirroring or Shadowing



○ Data Striping



○ Parallelism, Load Balancing, and Parity

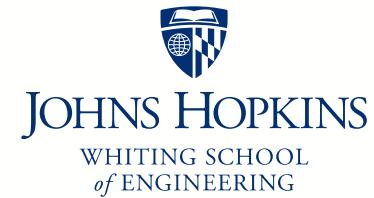
Redundant Array of Independent Disks (RAID) (cont.)

- RAID does NOT protect against power failures, operator mistakes, buggy RAID software, hardware failure, virus attacks on the storage system, inadvertently deleting files, or lightning hitting the building and causing a fire in the computer room.
- RAID is not a substitute for regularly scheduled maintenance and backup procedures.

Johns Hopkins Engineering

Principles of Database Systems

Module 11 / Lecture 2
File Structures and Indexes



Data Types, Files, and Records

- Data Types in a Record:
 - Data is stored in the form of records. Each record consists of a collection of related data values or items.
 - Each value corresponds to one of the basic data types: (Boolean, integer, string of characters, date) and advanced data types: (user-defined, pointers to external files, VARRAY, references, large objects (LOB), XML, and JSON).
- Files
 - A collection of records stored in a physical storage with the same format.
- Fixed/variable-length records.

Unspanned Versus Spanned Record Blocks

- An unspanned record block may contain multiple records and leave unused space at the end of the block.
- A spanned record block contains a pointer linking to the next block to store records.

Allocating File Blocks on Disk

- Continuous allocation: easy to access data blocks, hard to expand to new records.
- Linked allocation: easy to expand but is slow to access all data blocks.
- A combination of both allocates clusters (also called segments or extents) of consecutive blocks, and clusters are linked together.
- Indexed allocation: index blocks with indexes pointing to the actual data block.

Headers on File and Block

- File information is stored in a file header including the disk addresses of the file blocks, record format descriptions, and so on. Overhead causes less data storage space available in a data block.
- The purpose of a good file organization is to locate the needed block(s) by the DBMS with a minimal number of block transfers in order to improve the performance.

Operations on Files

- Basic operations include ‘insert’, ‘retrieve’, ‘update’, and ‘delete’ records or files.
- File organization concerns the organization of data file into records, blocks, and access structures. How are they stored in the physical storage on the disk and how are they interlinked?
- The access method uses different programs to perform basic database operations.
- Performance is the primary concern for file organization and access method.

Physical Database Design

- Database is implemented on the secondary storage
 - Database schema – database objects, constraints, triggers
 - File organizations – format and types
 - Indexes – performance
 - Size estimates
 - Security requirements – data and access control via system, application, or DBMS protection

Files of Unordered Records (Heap Files)

- A query for a record(s) uses a linear/sequential search if there is no access path on the file.
- A file of unordered fixed-length records using unspanned blocks and contiguous allocation, i -th record's location can be easily identified if the i value is known. However, the value does not relate to the search condition. Therefore, this type of file requires an access path using indexes.

Files of Ordered Records (Sorted Files)

- A query for a record is extremely efficient using a sorted key. Searching the next record is usually stored at the same block or the next block.
- A binary search based on the blocks rather than on the records.
- In general, a binary search reads for $\log_2 n$ blocks comparing $n/2$ for an unordered file.

Hashing File Organization

- Using the hashing function with the value of hash field to generate the address of the disk block (hash address) in which the record is stored. Only the equality search condition can be applied. The search is based on hash field of the file and is very fast.
- For example, employee_id ranges 1-1000, the hashing address = $f(\text{employee_id})$ where $f()$ = remainder ($\text{employee_id} / 1000$)

Hashing File Organization (cont.)

- There is a possibility that different values of a hash field have the same hashing address.
- The number of possible values of a hash field (hash field space) usually is much larger than that of available addresses for records (address space) to save unused spaces.
- A collision occurs when inserting a second record with a hashing address is used by another record.

Hashing File Organization (cont.)

- A good hashing function should distribute the records uniformly over the assigned spaces and minimize collisions.
- The process requires choosing a hashing function with a method for collision resolution.
- A suitable method for external hashing is the bucket technique (see Figure 16.9) and with chaining for bucket overflow caused by collisions (see Figure 16.10).

Hashing File Organization (cont.)

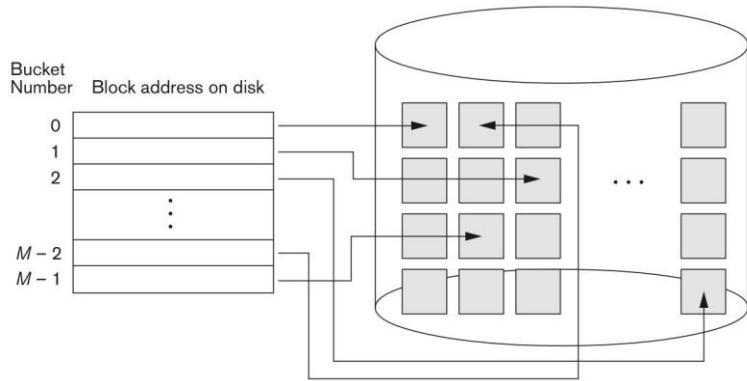


Figure 16.9 Matching bucket numbers to disk block addresses.

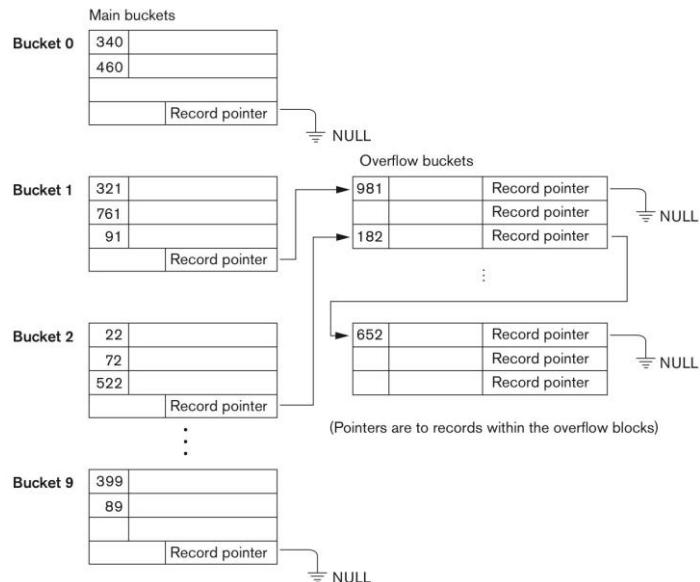
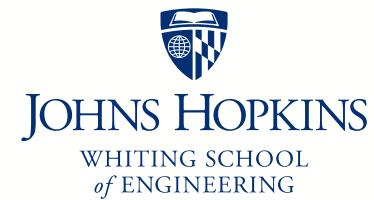


Figure 16.10 Handling overflow for buckets by chaining.

Johns Hopkins Engineering

Principles of Database Systems

Module 11 / Lecture 3
File Structures and Indexes



Index Structure for Files

- **Types of Single-level Ordered Indexes:**
 - The major advantage of an index is that it speeds up retrieval.
 - The update process is slowed down. When a new record is stored into a table, the index(es) for that record must also be added to the corresponding index file(s).

Primary Indexes

- **Primary Indexes:**
 - A primary Index is an index specified in the ordering key field of an ordered file of records (see Figure 17.1).
 - The ordering key field is called the primary key of a data file, and records can be uniquely identified.
 - An index file in which each entry (each record) consists of two values, the value of the primary key field for the first record in the block, and a pointer to that block.

Primary Indexes (cont.)

■ Primary Indexes (cont.):

Primary Index 

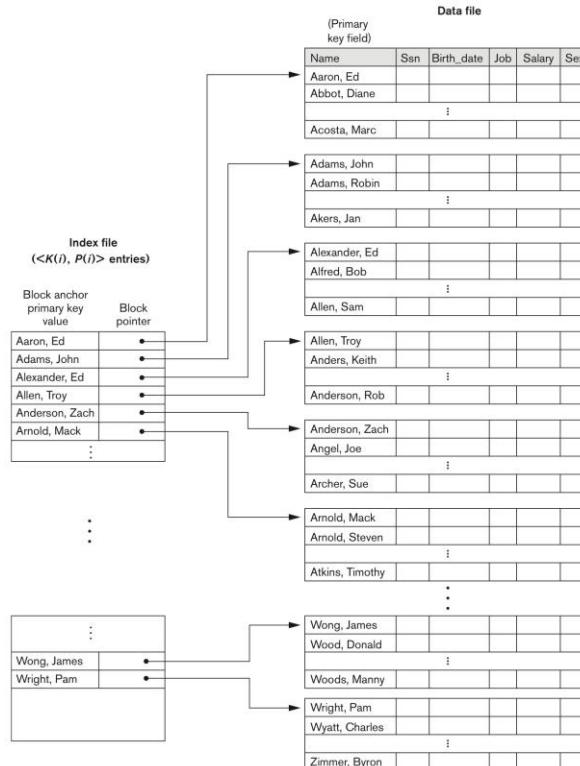


Figure 17.1 Primary index on the ordering key field of the file shown in Figure 16.7.

Primary Indexes (cont.)

- **Primary Indexes (cont.):**
 - The first record in each block of the data file is called the anchor record of the block.
 - A nondense index contains only one entry for each disk block of the data file. A dense index contains an entry for every record in the file.
 - A primary index is a nondense index.
 - Binary search can be applied on the index file (see Exercise 1 with 30K records – linear, binary and primary index searchers.)

Clustering Indexes

- **Clustering Indexes:**

- The file records are physically ordered in a nonkey field without distinct value for each record.
- The column of a table in a cluster is called the cluster field and a clustering index is created.
- Clustering index is a non-dense index. It has an entry for every distinct value of the indexing field (see Figure 17.2).
- Chaining data blocks resolves the inserting record problem (see Figure 17.3).

Clustering Indexes (cont.)

■ Clustering Indexes (cont.):

Clustering Index 

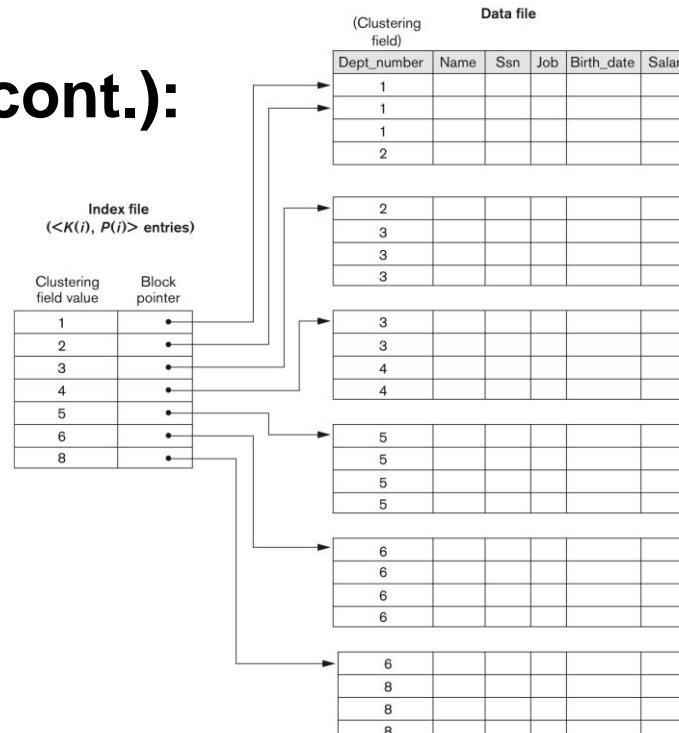


Figure 17.2 A clustering index on the `Dept_number` ordering nonkey field of an `EMPLOYEE` file.

Secondary Indexes

- **Secondary Indexes:**
 - A secondary index file provides alternative ways to access a file. It contains two fields. The first field is of the same data type as a non-ordering field of the data file, and is called an indexing field of the file. The second field can be a block pointer or a record pointer.
 - Many secondary indexes can be used to a data file.

Secondary Indexes (cont.)

■ Secondary Indexes (cont.):

A Dense Secondary Index 

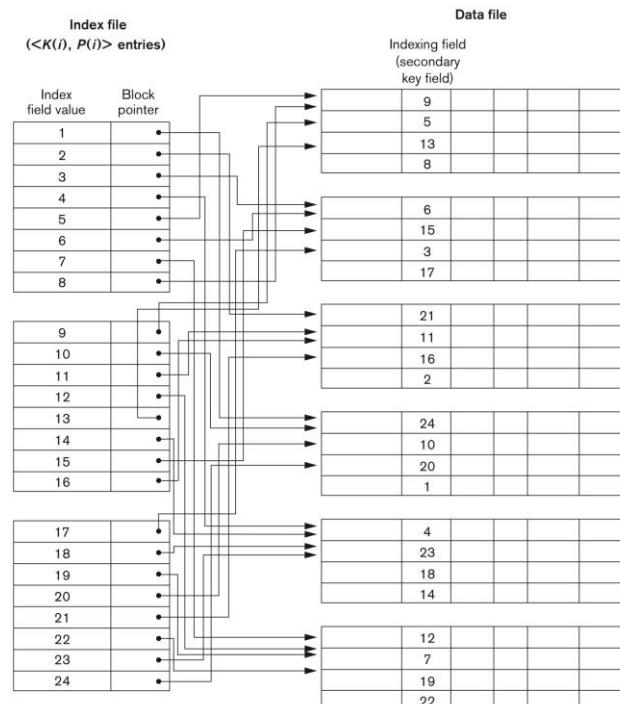


Figure 17.4 A dense secondary index (with block pointers) on a nonordering key field of a file.

Multilevel Indexes

- **Multilevel Indexes:**
 - Multilevel indexes are used to reduce the number of blocks accessed when searching indexes. The number of blocks needed to search a record is substantially reduced by a factor of $1/\text{fo}$, where fo is called *fan-out* of the multilevel index.
 - A multilevel index uses the key value of an ordered file as the first level of a multilevel index. The index for the first level is called the second level of index (see Figure 17.6).

Multilevel Indexes (cont.)

■ Multilevel Indexes (cont.)

A Two-level Primary Index

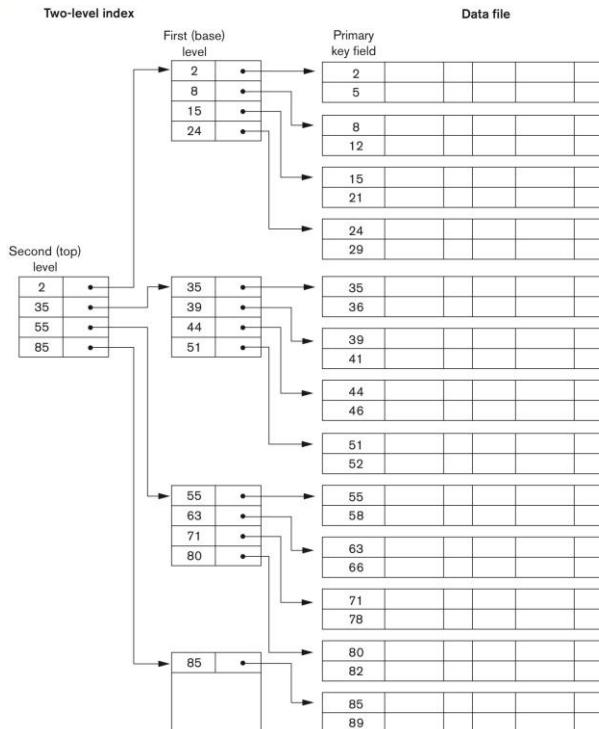
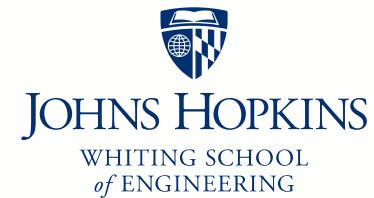


Figure 17.6 A two-level primary index resembling ISAM (indexed sequential access method) organization.

Johns Hopkins Engineering

Principles of Database Systems

Module 11 / Lecture 4
File Structures and Indexes



Dynamic Multilevel Indexes Using B-Trees

■ Dynamic Multilevel Indexes Using B-Trees:

- A tree consists of nodes. Each node has zero or more child nodes and one parent node, except for the root node (see Figure 17.7).

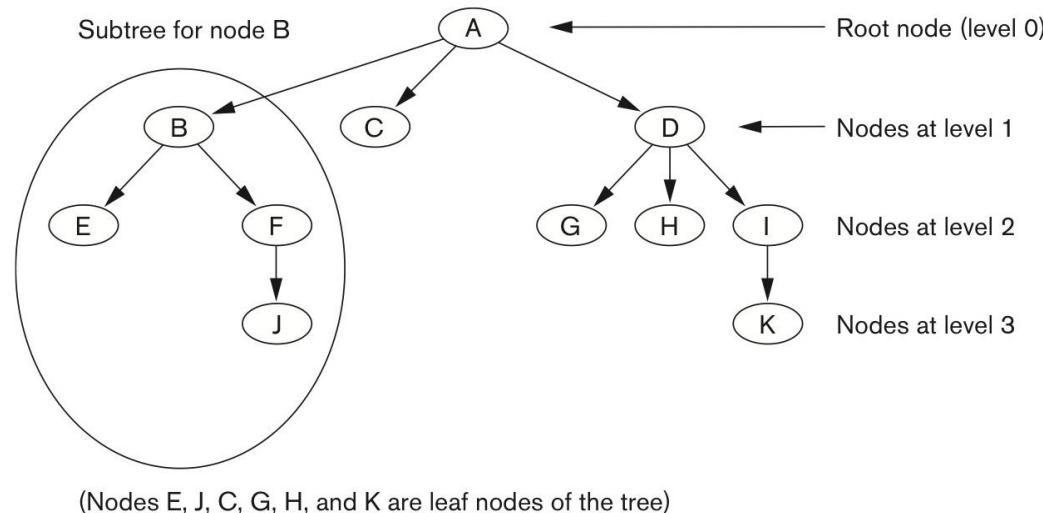


Figure 17.7 A tree data structure that shows an unbalanced tree.

Dynamic Multilevel Indexes Using B-Trees (cont.)

- **Dynamic Multilevel Indexes Using B-Trees (cont.):**
 - A search tree of order p is a tree such that each node contains at most $p-1$ search values and p pointers in the order $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$, where $q \leq p$. Each P_i is a pointer to a child node; each K_i is a search value. Two node constraints:
 - $K_1 < K_2 < \dots < K_{q-1}$
 - All values X in the subtree pointed at by P_i , $K_{i-1} < X < K_i$
 - B-tree structure is similar to the search tree with tree pointers P_i , search key fields K_i , and data pointers Pr_i .
 - B-tree has additional constraints to ensure that it is balanced.

Dynamic Multilevel Indexes Using B-Trees (cont.)

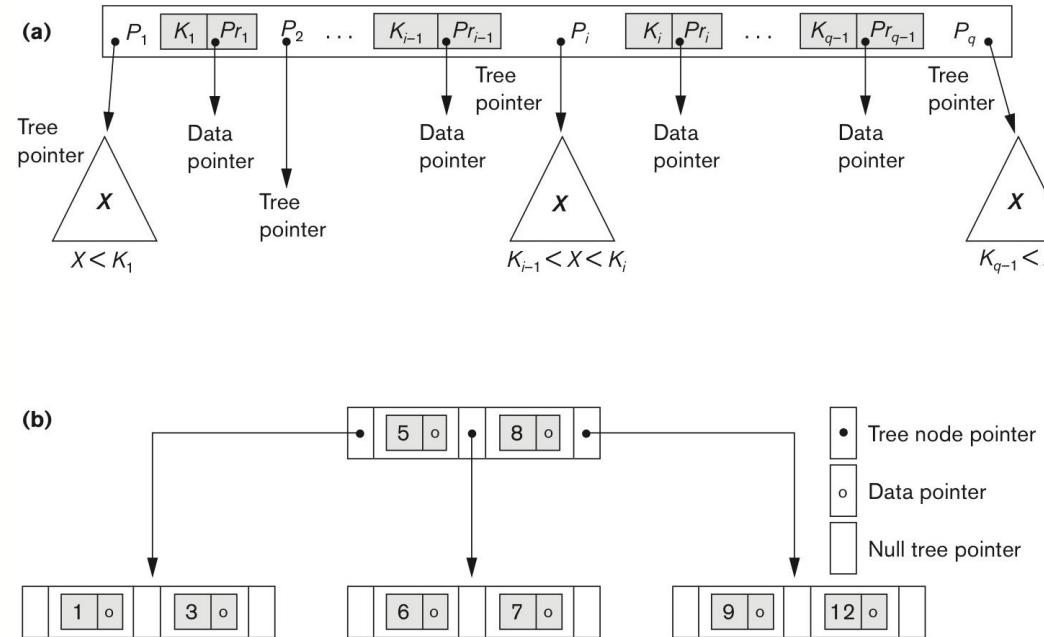


Figure 17.10 B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

Dynamic Multilevel Indexes Using B⁺-Trees

- **Dynamic Multilevel Indexes Using B⁺-Trees:**
 - Data pointers are stored only at the leaf nodes of the tree.
 - The leaf nodes have an entry for every value of the search field, along with a data pointer to the record if the search field is a key field.
 - The leaf nodes are linked together for ordered access on the search field to the records.

Dynamic Multilevel Indexes Using B⁺-Trees (cont.)

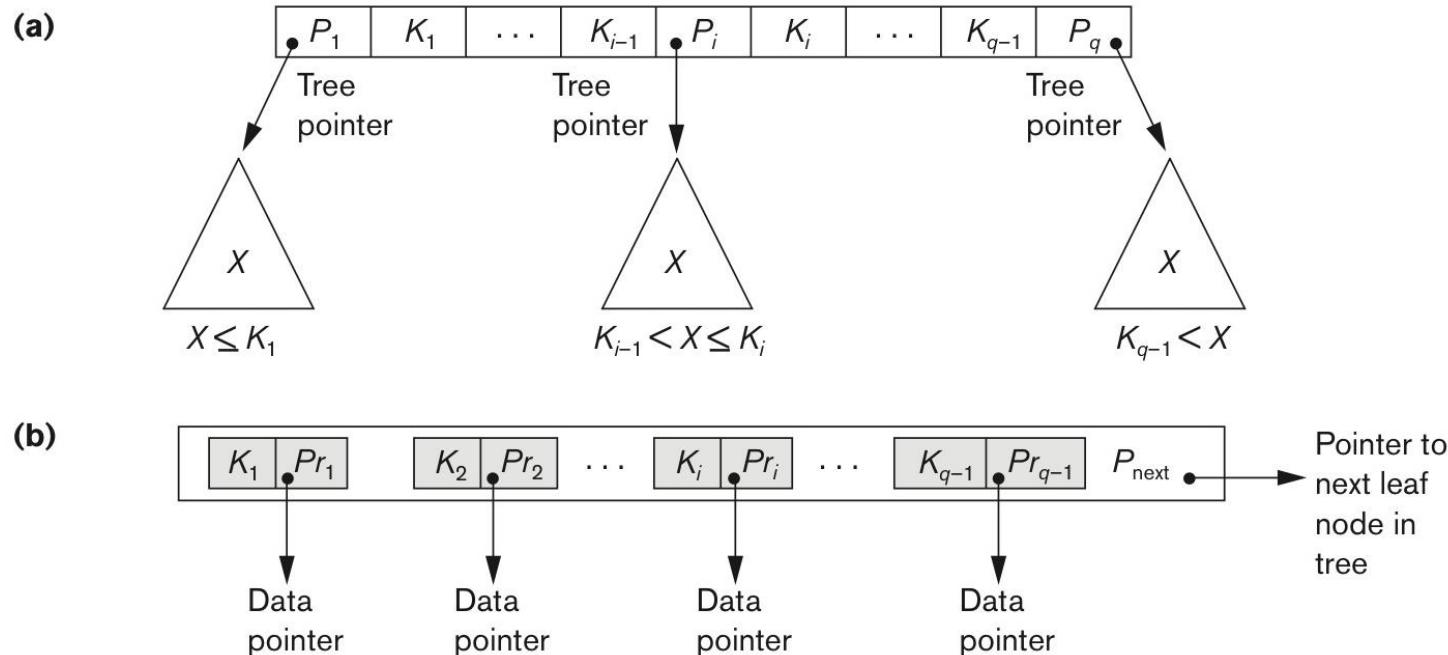


Figure 17.11 The nodes of a B+-tree. (a) Internal node of a B+-tree with $q - 1$ search values. (b) Leaf node of a B+-tree with $q - 1$ search values and $q - 1$ data pointers.

Dynamic Multilevel Indexes Using B+-Trees

(cont.)

- **Dynamic Multilevel Indexes Using B+-Trees (cont.):**
 - The internal nodes include search values and tree pointers without any data pointer to build other level(s) of multilevel indexes. More entries can be packed into an internal node of a B+-tree than for a similar B-tree (see Examples 6 and 7).
 - Because there is no single storage structure that is optimal for all applications, B-tree and B+-tree provide the best all-around performance.

Benefits and Trade-offs of Creating Indexes

- Overhead involved in maintenance and use of secondary indexes that has to be balanced against performance improvement gained when retrieving data.
- Create an index on a column, if:
 - The column is used frequently in WHERE clauses or in a join condition.
 - Every value within the column is unique.
 - The column contains a wide range of values (high cardinality column).

Benefits and Trade-offs of Creating Indexes (cont.)

- A function-based index is an index built on an expression.
- A function-based index precomputes the value of a computationally intensive function and stores it in the index. An index can store computationally intensive expression that you may access often

```
CREATE INDEX Idx ON Example_tab(Col_a + Col_b);  
SELECT * FROM Example_tab WHERE Col_a + Col_b < 10;
```

- The cost-based optimizer may prefer to scan a table instead of using the indexes.

Benefits and Trade-offs of Creating Indexes (cont.)

- A composite key (multicolumn index) is a primary key that consists of more than one column.
 - The order of the columns in a composite index is important and it also depends upon how your queries are written.
 - A composite index has the left-most (leading) column that occurs most often in the queries.
 - If the leading column is more selective, it can reduce the query cost.

```
SQL> count(*), count(distinct col_1), count(distinct col_2), count(distinct col_3)
2  FROM test_table;
```

Benefits and Trade-offs of Creating Indexes (cont.)

- **Do not create indexes, if:**

- The table is small.
- Most queries are expected to retrieve more than 10-15% of the rows.
- The table is updated frequently.
- A similar index is created (e.g., Index1 on (C1) and Index2 on (C1, C2)).

Good index design balances query performance and system maintenance cost.

Types of database applications

- **Online Transaction Processing (OLTP)**
 - May involve many simple DMLs
 - Determine required indexes
 - Minimize the number of OLTP indexes
 - May have complex queries for statistical reports
 - Use EXPLAIN PLAN to show how a statement will be processed (by the optimizer)

Types of database applications (cont.)

■ Data Warehouse

- Consolidate view of enterprise data from various databases
- May design and use to support decision making through Online Analytical Processing (OLAP analysis) and data mining
- Is optimized for reporting and analysis
- May involve complex queries for processing a large amount of data
- Use tools to analyze user queries

Johns Hopkins Engineering

Principles of Database Systems

Module 11 / Lecture 5
File Structures and Indexes



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Oracle Index Implementations

- **Using B⁺-tree (B^{*}-tree)**
 - An index file in which each entry (each record) consists of two values, a key value (a data value) and a pointer (ROWID).
 - Every row in a table of an ORACLE database is assigned a unique ROWID (a pseudo-column) that corresponds to the physical address of a row's row piece (the initial row piece if the row is chained among multiple row pieces).

Oracle Index Implementations (cont.)

- **ROWID Format with a *10-byte* address:**

OOOOOO - Data Object #

FFF - Relative File #

BBBBBB - Block #

SSS – Slot #

A base-64 encoding with the characters A through Z, a through z, 0 through 9, + and /.

Example: Show ROWID and ename in emp table

```
SQL> SELECT ROWID, ename FROM emp;
      ROWID          ENAME
-----+
AAAVJAAEAAABEAAA SMITH
AAAVJAAEAAABEAAB ALLEN
AAAVJAAEAAABEAAC WARD
...
D Obj#RF#Blk #SL#
```

Index-organized Table

- **Index-organized Table:**

- An *index-organized table* (IOT) with the data for a table is held in its associated index.
- IOT provides faster key-based access to table data for queries.
- The data rows are built on the primary key (must be specified) for the table, and each B*-tree index entry contains:
`<primary_key_value, non_primary_key_column_values>`

Bitmap Index

- **Bitmap Index:**
 - Indexes designed for use in large, slow-moving systems (e.g., data warehouse features for decision support) including columns with low distinct values.

Query Example:

Tell me about the customers living in the DC suburbs, who own 2 cars, have 3 children, live in a 4-bedroom house.

- Any single condition in this search may return a very large number of rows, no need to create B*-tree indexes on any specific combination of columns.
- Full table scan.

Bitmap Index (cont.)

- **Bitmap Index (cont.):**
 - How bitmap index work:
 - In a *bitmap index*, a bitmap for each key value is used instead of a list of rowids.
 - A bitmaps for Cars (0, 1, 2, ...) and Children (0, 1, 2, 3, ...), and Bedrooms (0, 1, 2, 3, 4, ...) respectively.
 - Using the Bitwise “AND” operator to filter qualified rows (key values).
 - Bitmap indexes are particularly useful for queries that contain [NOT] EXIST, UNIQUE, or GROUP BY on low cardinality columns.

Bitmap Index (cont.)

- **Bitmap Index (cont.):**
 - Benefits for using bitmap index:
 - Reduced response time for large classes of ad hoc queries.
 - A substantial reduction of space usage compared to other indexing techniques.
 - Dramatic performance gains even on very low end hardware.
 - Bitmap indexes are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data.

Table and Index Partitions

- Very large tables and indexes can be partitioned into smaller, more manageable pieces.
 - Improve performance with focused data access, and speed up data access from minutes to seconds.
 - Increase availability to critical information.
 - Improve manageability over data.

Table and Index Partitions (cont.)

- Proper information lifecycle management
 - As data ages, activity of the data declines and overall volume grows (e.g., business sales, bank transactions)
 - Current month data – most active data set (small size) stored in high performance storage
 - Previous month data – less active data set (medium size) stored in low cost storage
 - Previous year data – historical data set (large size) stored in cheap cost storage
 - Previous years data – archive data set (huge size) stored in offline storage
- Partitioning by range via a partition key (e.g., number or date), by list of values, by hash

Table and Index Partitions (cont.)

- Multiple columns are allowed.
... **PARTITION BY RANGE** (year, month, day)
(PARTITION p1 VALUES LESS THAN (2016, 12, 31) TABLESPACE ts201612),
PARTITION p2 VALUES LESS THAN (2017, 12, 31) TABLESPACE ts201712),
.....,
PARTITION pOutRange VALUES LESS THAN (MAXVALUE, MAXVALUE, MAXVALUE)
TABLESPACE tsOutRange);
- A partitioned table can have partitioned or non-partitioned indexes.
- The major reason for using range partitioning is the **tremendous performance benefit** (e.g., query, parallel DML, import/export).

Table and Index Partitions (cont.)

- Backup and recovery process increases overall availability.
- DBMS handles the access methods to reduce amount of touched data (IO reduction) to speed up response time or improve throughput.
- Partitions are transparent to developers and database applications.
- Partitions can have local or global indexes.
- Partitions are useful for very large databases and high performance and availability requirements.

Performance tuning for VLDB

- A CPU bound database server (examining top 5 timed events)
 - Not easy to tune computationally intensive applications
 - Using faster CPU and parallel processing
- An I/O bound database server
 - Add more RAM buffers for database usage and SQL tuning are the best way to reduce disk I/O
- Using 64-bit technology
 - Improving RAM addressing; Faster CPU and file I/O; High parallelism with multiple CPUs
- NoSQL for big data with performance, scalability, flexible schema advantages